МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

филиал ФГБОУ ВО «РГГМУ» в г. Туапсе

Кафедра «Экономики и управления на предприятии природопользования»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(бакалаврская работа)
по направлению подготовки 09.03.03 Прикладная информатика
(квалификация – бакалавр)

На тему « Web-приложение с элементами геоинформационных технологий»

Исполнитель Лебедев Давид Игоревич

Руководитель к.г.н., доцент, Полупанов Владимир Николаевич

«К защите допускаю»

Руководитель кафедры _____

кандидат экономических наук

Майборода Евгений Викторович

«23» января 2024 г.

Филиал Российского государственного гидрометеорологического университета в г. Туапсе
НОРМОКОНТРОЛЬ ПРОЙДЕН
«19» января 2024 г.
_____ Мартынова М.В
ПОДПИСЬ РАСШИФРОВКА ПОДПИСИ

Туапсе
2024

ОГЛАВЛЕНИЕ

Введение

Необходимость в оцифровке (digitizing) рельефа земной поверхности и, в частности, дна водных объектов возникает чаще всего в связи с необходимостью восстановления поля высот (глубин) по данным точечных измерений высот (глубин), получаемых в полевых условиях: геодезические и батиметрические площадные съёмки с участием и без участия людей (беспилотные измерения). Процедура оцифровки состоит из следующих основных этапов:

- перенесение точек с измерениями высот с бумажной карты (планшета) на электронную карту или (и) нанесение (добавление) точек с измерениями из других источников;
- интерполяция высот в узлы регулярной сетки и отображение в виде, удобном для визуального восприятия (цветовых карт, изолиний), редактирование исходных данных;
- формирование выходного файла с результатами оцифровки для передачи (экспорта) в другие программные системы.

Для оцифровки используются компьютерные приложения с элементами геоинформационных технологий, чаще всего реализованные в составе геоинформационных систем (ГИС).

ГИС – сложные универсальные программные системы, эксплуатация которых связана с большими трудозатратами, тогда как для небольших участков иногда достаточно реализовать лишь процедуру оцифровки, без избыточного использования всей функциональности ГИС.

Актуальность настоящей работы обусловлена тем, что отсутствует легкодоступное и бесплатное приложение для оцифровки, способное выполняться на бюджетных компьютерах и планшетах, в том числе в полевых условиях.

В то же время представляется, что автоматизацию процедуры оцифровки для сравнительно небольших участков наиболее эффективно можно

осуществить, используя технологии Web.

Объектом настоящего исследования являются технологии Web и ГИС, необходимые для реализации оцифровки.

Предмет исследования – возможности языка Javascript для разработки приложения оцифровки в среде клиента.

Проблемность работы связана с тем, что обычно оцифровка реализуется на языках, традиционно применяемыми для «тяжёлых вычислений» на стороне сервера (в «десктопном» варианте - компьютера), таких как Фортран, C, C++ и других, производных от C.

Основным же языком в Web является Javascript, первоначально предназначенный для программирования на стороне клиента в Web-дизайне, но не для реализации вычислительных задач.

Однако Javascript оказался чрезвычайно быстро развивающимся языком, впитавшим практически все технологии других Си-подобных языков. К тому же сейчас его можно использовать как на стороне клиента, так и на стороне сервера.

Следует отметить также стремительное повышение быстродействия клиентских Javascript-движков, благодаря чему появляется всё больше приложений, реализующих вычислительные задачи на стороне клиента.

Представляется, что сегодня Javascript недооценён с точки зрения возможностей для реализации задач вычислительной математики.

Целью бакалаврской работы является разработка приложения для оцифровки батиметрии с использованием Web-технологий.

Для достижения поставленной цели необходимо решить задачи:

- рассмотреть средства оцифровки, используемые в ГИС;
- проанализировать возможности Web для реализации оцифровки;
- предложить проект и осуществить реализацию Web-приложения оцифровки;
- показать работоспособность Web-приложения на примере.

Следует подчеркнуть: Web-приложение должно быть открытым («Open

Source») и бесплатным («Free Software»).

Реализация оцифровки в Web может послужить ещё одним примером приложения, в котором для программирования задач вычислительной математики используется Javascript.

1 Оцифровки в ГИС и Web

1.1 Оцифровка в ГИС

Варианты решения задачи оцифровки рассмотрим на примере ГИС QGIS [20]. В этой бесплатной («Free Software») и свободно распространяемой («Open Source») системе, имеются инструменты оцифровки, свойственные современным ГИС. При оцифровке в ГИС необходимо пройти этапы:

1) привязка растрового изображения карты к базовой системе координат;

2) создание точечного слоя с координатами и высотами (глубинами) по имеющейся батиметрии на карте;

3) интерполяция высот (глубин) в узлы регулярной сетки и отображение в виде, удобном для визуального анализа и редактирования точечного слоя;

4) повторение этапов 2 и 3 при необходимости;

5) сохранение результатов оцифровки в формате, пригодном для экспорта в другие приложения, для которых требуются результаты оцифровки.

1.1.1 Привязка изображения карты

Процедура привязки состоит в установлении однозначного соответствия между координатами изображения и системой координат (СК), к которой привязывается изображение [1,2,6]. На этапе привязки также устраняются искажения, допущенные при подготовке цифрового изображения карты (деформации бумаги, искажения скана, фотоснимка). Для устранения искажений используются так называемые «трансформации» – математические методы преобразования искажённого изображения к исходному неискажённому.

Рассмотрим процедуру привязки на примере изображения карты г. Кинешма с окрестностями. Это среднемасштабная топографическая карта в проекции Pulkovo 1942/Gauss-Kruger zone 8, СК EPSG:28408 [15]. Масштаб

карты 1: 100000, то есть в 1см на карте – 100000 см на местности (или в одном см – один км, такие карты называют «километровками»).

Координаты опорных точек, выбранных для привязки, приведены в таблице 1.1, а расположение - на рисунке 1.1. Нумерация точек соответствует индексам, присваиваемым по умолчанию при привязке. Координаты точек в «родной» проекции карты задаются в метрах, но на «бумажной» карте нанесены также шкалы, позволяющие снять географические координаты в формате GG MM SS. Последние предполагается использовать для контроля в проекции с географическими координатами (например, WGS-84), сравнивая их с координатами, которые можно снять с Яндекс Карт, Google Карт, Google Earth и прочих сервисов. Для этого, координаты, снятые с бумажной карты, переведены в десятичные градусы с использованием онлайн-сервиса «Геокалькулятор – пересчёт координат» [21].

Таблица 1.1 — Координаты опорных точек: прямоугольные (метры) и географические (GG MM SS и GG.000000)

Точка	X, метры	Долгота	Y, метры	Широта
0	8321000	42°00′00″E (42.000000°)	6398200	57°40′00″N (57.666667°)
1	8350900	42°30′00″E (42.500000°)	6397000	57°40′00″N (57.666667°)
2	8349200	42°30′00″E (42.500000°)	6359950	57°20′00″N (57.333333°)
3	8319200	42°00′00″E (42.000000°)	6361000	57°20′00″N (57.333333°)
4	8322000	42°00′57″E (42.015700°)	6396000	57°38′51″N (57.647500°)
5	8348000	42°27′28″E (42.457600°)	6396000	57°39′24″N (57.656600°)
6	8348000	42°27′28″E (42.457600°)	6362000	57°21′07″N (57.352000°)
7	8322000	42°02′25″E (42.040100°)	6362000	57°20′31″N (57.341900°)
8	8330600	42°10′35″E (42.176470°)	6372700	57°26′30″N (57.441500°)
9	8336500	42°15′50″E (42.263910°)	6388900	57°35′20″N (57.589110°)

Рисунок 1.1 — Изображение карты, используемой для привязки в качестве подложки (маркером отмечены опорные точки)

После привязки и обрезки рамки карта в СК EPSG:28408 будет иметь вид (с добавлением слоя из Google Hybrid).

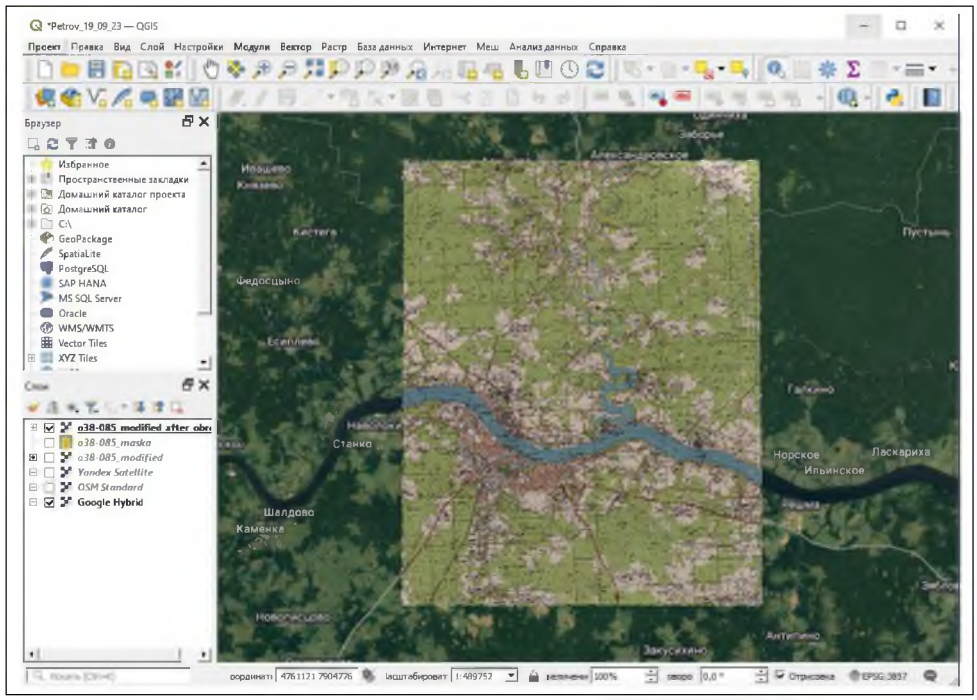

Рисунок 1.2 — Карта в СК EPSG:28408.

1.1.2 Создание точечного слоя

Работать будем с участком карты, входящим в шесть квадратов покрывающих Горьковское водохранилище: 7436, 7438,7440,7236,7338,7240.

Рисунок 1.3 — Шесть квадратов карты для оцифровки в СК

Нанёсём на карту вначале точки изогипсы на суше вдоль бровки поймы (высоту которой примем за 84 м БС), а затем – точки береговой линии водохранилища и, наконец – изобаты. Высоту береговой линии будем считать равной 81 м БС (при статичном уровне воды это значение постоянно для всей береговой линии). При этом сразу же заносим значения в глубинах: - 3 м для бровки (81-84) и 0 м для береговой линии.

По завершению оцифровки точечный слой может иметь следующий вид (на голубом фоне полигонального слоя).

Анализируя проделанную работу следует отметить большую трудоёмкость оцифровки в QGIS. Все точки приходится заносить вручную, тогда как в других системах, например в Surfer [5], имеется возможность автоматизировать занесение точек изолиний. Кроме этого, при ручном занесении точек изолинии трудно соблюдать равномерность расположения

точек, что проявится далее, при интерполяции глубин по площади.

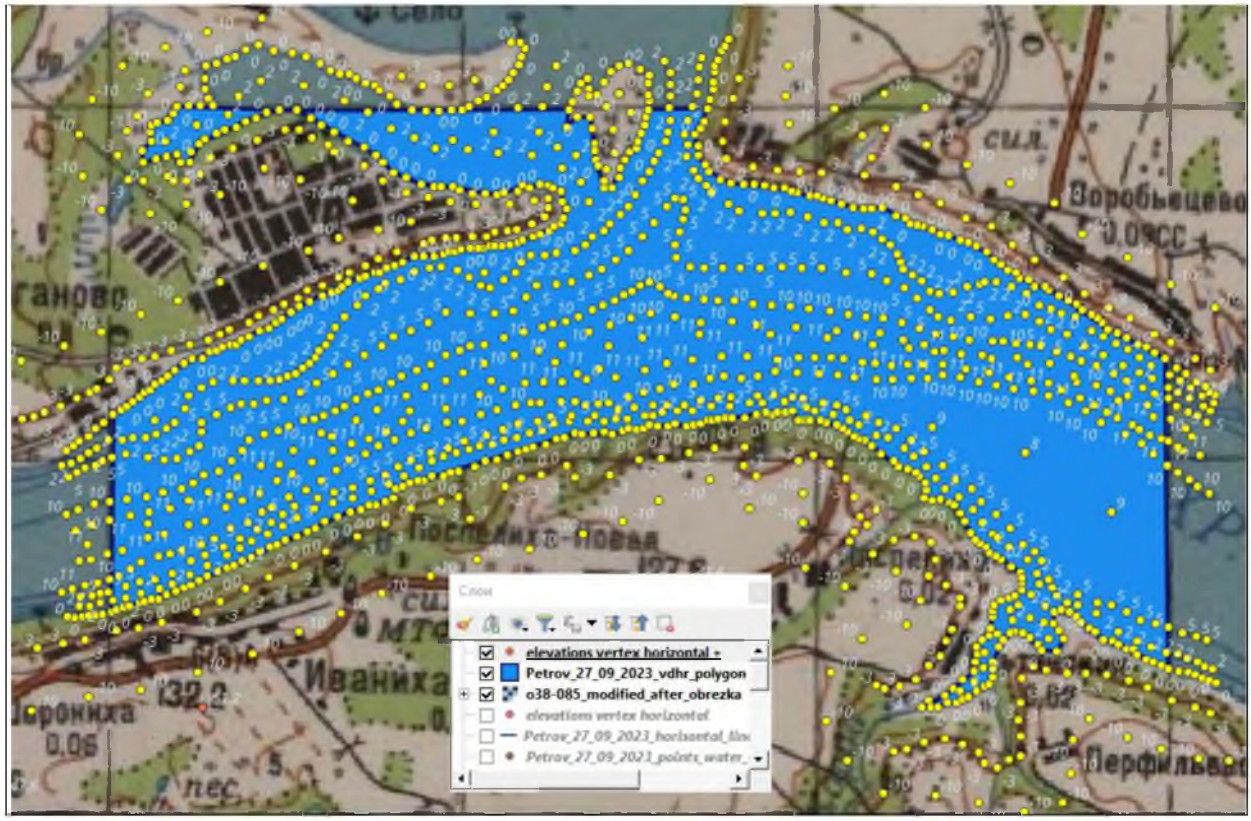

Рисунок 1.4 — Точечный слой – результат оцифровки глубин в QGIS

Отмеченные недостатки оцифровки в QGIS необходимо будет учесть при разработке Web-приложения.

1.1.3 Интерполяция в узлы регулярной сетки

Качество полученных данных, размещённых в точках, нерегулярно расположенных по площади участка, трудно оценить визуально. Поэтому глубины из полученных точек интерполируют по площадной (двумерной) сетке, достаточно подробной для последующего визуального анализа на предмет ошибок, возможно полученных при оцифровке. Для двумерной интерполяции в QGIS можно использовать несколько модулей. Некоторые встроены в QGIS, другие доступны в QGIS через SAGA GIS и GRASS GIS, а также через доступ к геоалгоритмам GDAL/OGR [16,20]. Разновидностей алгоритмов, используемых в этих модулях достаточно много, но можно перечислить следующие основные

типы:

- полиномиальная интерполяция;
- интерполяция сплайнами;
- крайкинг-интерполяция;
- интерполяция обратным взвешиванием расстояний (ОВР).

Мы будем использовать последний метод, наиболее популярный при обработке глубин.

Метод ОВР или IDW (англ. Inverse Distance Weighting) заключается в том, что при вычислении интерполируемого значения Z происходит взвешивание точек с весами (ω_i) таким образом, что влияние известного значения (x_i) в i-той точке затухает с увеличением расстояния (d) до точки (x), значение которой надо определить [19]:

$$Z(x) = \sum_{i=1}^{n} \frac{\omega_i(x) z_i}{\sum_{j=1}^{n} \omega_j(x)}, \quad \omega_i(x) = \frac{1}{d(x,x_i)^{\alpha}}. \tag{1.1}$$

Чаще всего используется значение α = 2, то есть веса уменьшаются обратно пропорционально квадрату расстояния до влияющих точек.

Основной недостаток метода ОВР состоит в том, что при его использовании соотношение исходных точек может быть таким, что приводит к чрезмерному выделению локальных особенность рельефа дна вокруг влияющих точек (эффект «бычьих глаз»). В связи с этим разработаны методы, улучшающие качество интерполяции методом ОВР [17,18].

Следует отметить, что в QGIS интерполяция проводится с преобразованием в растр, где каждая точка растра – это пиксель дисплея.

ОВР-интерполяция – наиболее часто используемый метод при интерполяции глубин, в основном, из-за вычислительной устойчивости метода [19]. При отборе влияющих точек их размещение должно быть равномерным вокруг точки с определяемым значением. В этом случае происходит

интерполяция, иначе, при расположении точек с одной стороны – экстраполяция, что по определению хуже интерполяции.

Важность равномерного размещения влияющих точек необходимо учесть при реализации OBP-метода в Web-приложении.

Итак, возьмём оцифрованный точечный слой глубин (см. рисунок 1.4) и создадим растровый слой глубин двумерной интерполяцией методом OBP (Растр – Анализ – Интерполяция (обратно-взвешенное расстояние)

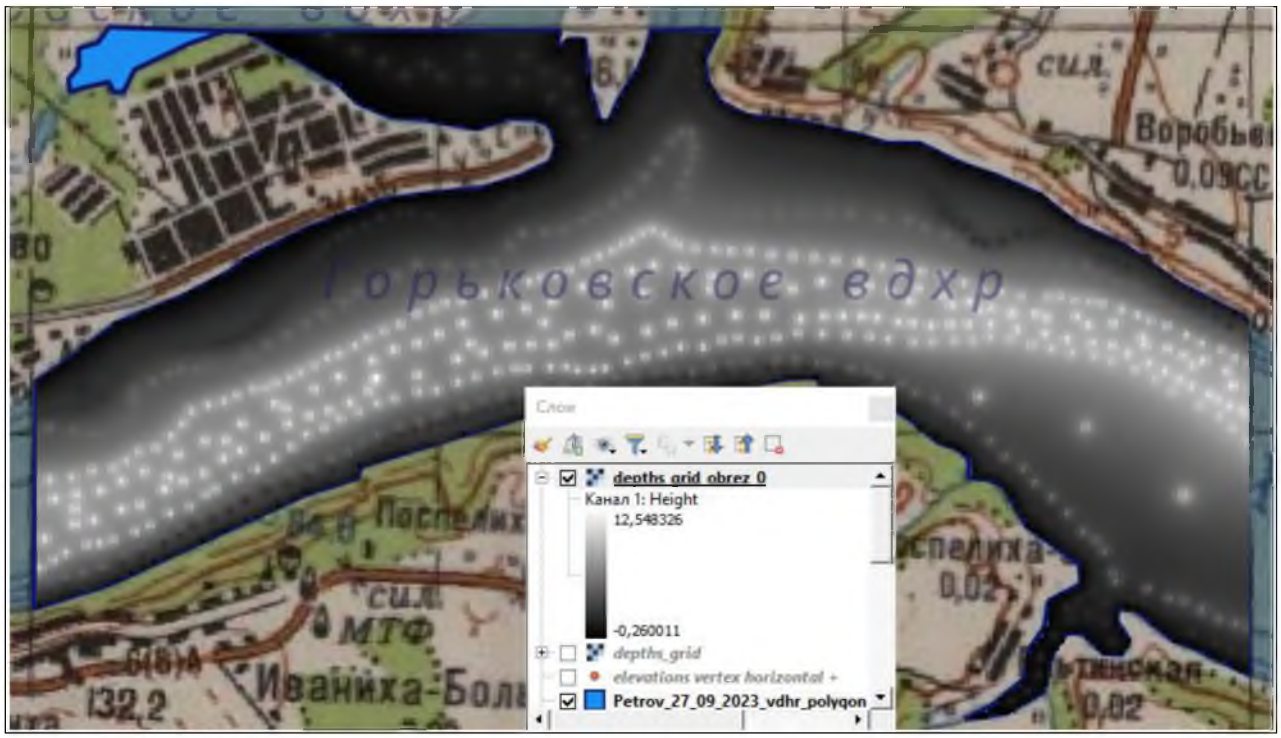

Рисунок 1.5 — Создаем растровый слой глубин OBP-методом по данным точечного слоя (по результатам оцифровки глубин)

Количество влияющих точек возьмём равным 6-ти. Этот параметр подбирается экспериментально: при увеличении получаемый растр будет более сглаженным.

Для улучшения визуального восприятия изменим окраску растра с использованием более подходящей палитры.

Вместо цветовой палитры для растра можно выбрать стиль «Изолинии». Тогда два слоя: с цветовой палитрой (нижележащий) и слой с изолиниями (вышележащий), возможно более удобны для визуального анализа.

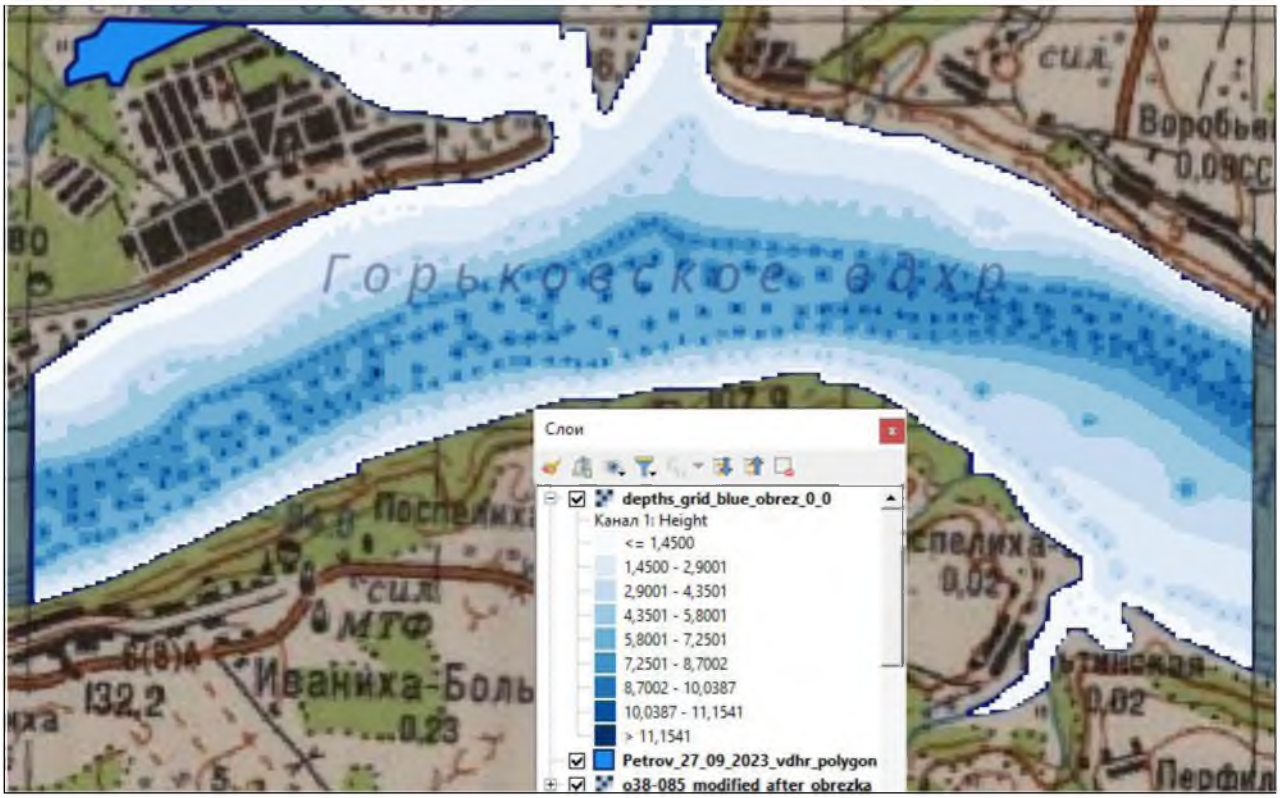

Рисунок 1.6 — Растровый слой глубин с изменённой цветовой палитрой

Вместо цветовой палитры для растра можно выбрать стиль «Изолинии». Тогда два слоя: с цветовой палитрой (нижележащий) и слой с изолиниями (вышележащий), возможно более удобны для визуального анализа.

Следует заметить, что стиль «Изолинии» – это растровый слой, в отличие от векторного слоя изолиний, создаваемого инструментом: Растр – Извлечение – Создать изолинии, с соответствующими различиями по манипуляции изолиниями для растровых и векторных слоёв.

В результате мы видим цветовую модель глубин. При наличии грубых ошибок можно сделать исправления в точечное слое и повторить отображение глубин в виде цветного растра. Можно отметить наличие артефактов вокруг точек с известными глубинами. Имеющиеся инструменты сглаживания поля глубин слишком искажают морфометрические характеристики водохранилища.

К сожалению в QGIS нет возможности менять размер ячейки растра. С каждым пикселем растра связано значение глубины (Height). Растр в QGIS не имеет атрибутивной таблицы со значениями глубин, имеющимися у векторного

точечного слоя и на его основе нет возможности быстро и просто рассчитать количественные характеристики, которые можно было использовать для подтверждения адекватности сравнения результатов, как это можно делать в других системах.

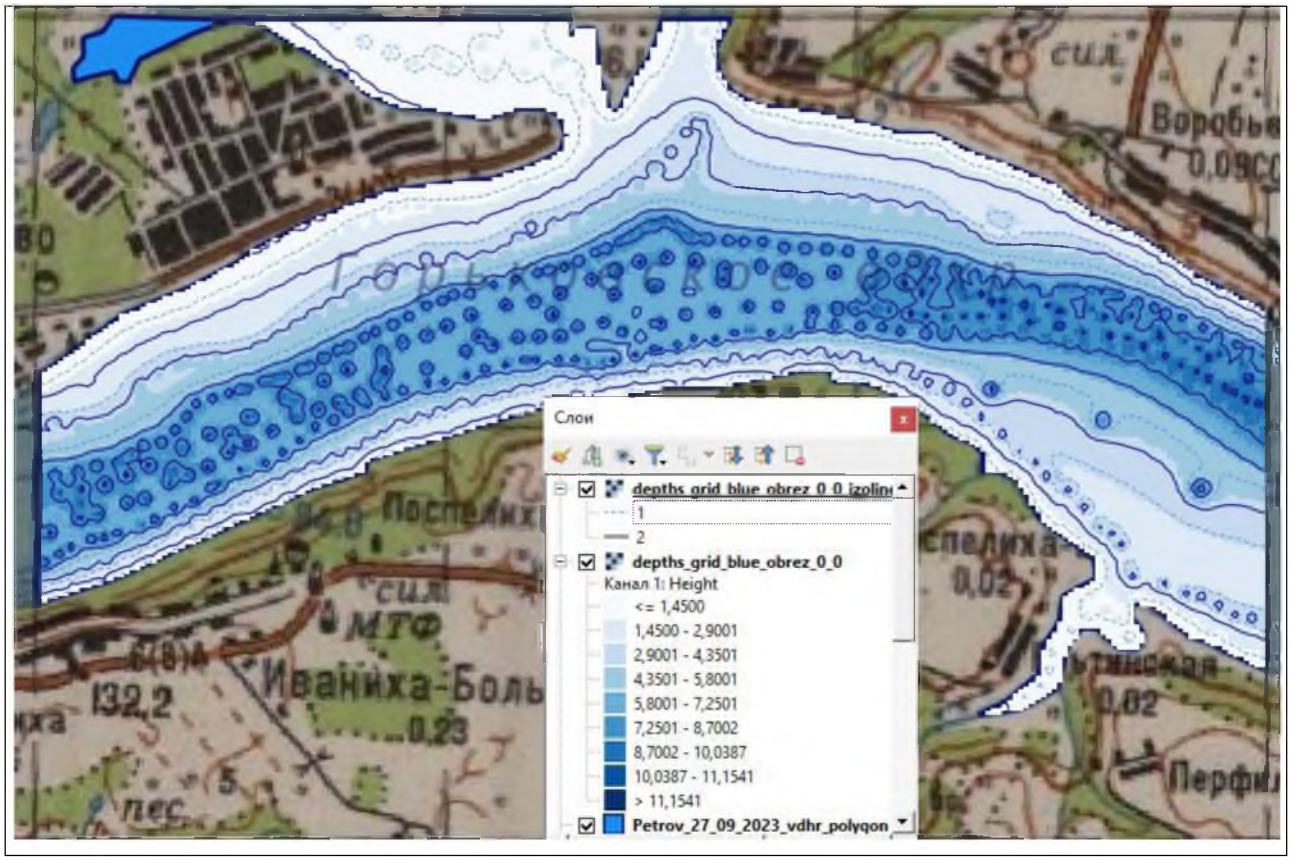

Рисунок 1.7 — Два растровых слоя глубин: с цветовой палитрой и с изолиниями

Например, в пакете Surfer [5], можно получать площади и объёмы для различных глубин водоёма, графики глубин по разрезам и т.п. Но для этого ОВР-метод надо приспособить для интерполяции не в узлы растра, а в клетки клеточной области (пикселя с регулируемым размером и с атрибутивной таблицей), присваивая им значения глубин. Регулируемый растр предусмотрен, например, в инструменте Spatial analist пакета arcGIS [23]. Изменением размера квадрата можно было бы изменять степень усреднения по площади и таким образом избавляться от артефактов, вызываемых свойством ОВР-метода отдавать или убирать преимущество локальным особенностям рельефа вокруг влияющих точек.

Возможность ОВР-интерполяции в регулируемые клетки необходимо предусмотреть при разработке Web-приложения.

Впрочем, к настоящему времени появились модули (встроенные, как стандартные инструменты в QGIS версии 3.28 LTR), и по полученному растру можно получить площадь зеркала, объём водоёма и другие характеристики. Для этого можно использовать инструменты: Анализ данных – Инструменты анализа – Анализ растров – Объём растровой поверхности, а также: … - Анализ растров – Статистика растрового слоя.

Итого, имеем следующие характеристики растрового слоя глубин:

- объём водоёма: 40700185,2 $м^3$;
- площадь зеркала: 9154707,9 $м^2$.
- максимальное значение глубины: 12,55 м;
- среднее значение глубины: 4,42 м;
- количество пикселей: 19342;
- размеры пиксела (высота x ширина): 17,6475 м x 26,8201 м.

Эти характеристики впоследствии используем для сравнения со значениями, которые будут получены в Web-приложении.

1.1.4 Сохранение и экспорт-импорт результатов

Результаты оцифровки обычно являются входными данными для программ, например, для гидродинамического моделирования и т.п. В настоящее время универсальным форматом для обмена геоданными является GeoGSON [25]. GeoJSON — открытый формат, предназначенный для хранения географических структур данных, основанный на JSON [26].

JSON (англ. JavaScript Object Notation— текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. Формат JSON был разработан Дугласом Крокфордом. Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается

независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

Поскольку формат JSON является подмножеством синтаксиса языка JavaScript, то он может быть быстро десериализован встроенной функцией JSON.parse().

JSON-текст представляет собой (в закодированном виде) одну из двух структур данных, поддерживаемых любым языком программирования: 1) набор пар ключ: значение; 2) упорядоченный набор значений (во многих языках это реализовано как массив, вектор, список или последовательность).

В формате GeoJSON геометрические объекты определены как: Point, LineString, Polygon, MultiPoint, MultiLineSring и MultiPolygon.

Пример первой структуры (ключ: значение) в формате JSON с элементами GeoGSON может иметь вид:

```
{
    "coordinates": [42.66667, 53.21555]
    "tipe": "point",
}
```

Массив точек в формате JSON можно реализовать в виде:

```
{ "type": "MultiPoint",
    "coordinates": [
        [42.667,53.216],[42.67,53.16],[42.77,53.36],[ 42.67,53.22]
    ]
}
```

Здесь "type" определяет тип геометрии, а "coordinates" представляет координаты точки в формате [долгота, широта].

"propeeties" – объект (в смысле JS-объекта), содержащий дополнительные свойства точечного объекта.

Чтобы описать объект вместе с его атрибутами (в нашем случае – с глубиной, в GeoJSON сделали сущность Feature, которая обязательно должна

иметь поля: type – тип (Feature); geometry – описание геометрии; properties – атрибуты объекта (в виде JS-объекта).

И, наконец, для описания слоёв введена сущность FeatureCollection. Это сущность самого верхнего уровня и должна иметь поля: type – тип (FeatureCollection); features - массив объектов Feature.

Система координат может быть описана путём указания имени. В этом случае значение свойства «type» должно содержать строку «name». Значение свойства «properties» должно представлять объект, содержащий свойство «name». Значением свойства «name» должна быть строка, определяющая систему координат. Предпочтительней использовать идентификаторы проекций OGC CRS URN вместо таких идентификаторов, как «EPSG:4326» [15].

Чтобы включить информацию о диапазоне координат для геометрий, элементарных объектов и коллекций элементарных объектов используется свойство «bbox» (ограничивающий прямоугольник) объекта GeoJSON. Значение данного свойства должно быть массивом размерности 2*n, где n – размерность входящих в объект геометрий и содержать минимальные и максимальные значения координат всех координатных осей. Порядок осей при описании ограничивающего прямоугольника соответствует порядку осей, используемых при описании геометрий. Кроме того, предполагается, что система координат ограничивающего прямоугольника соответствует системе координат объекта GeoJSON, свойством которого она является.

В QGIS имеются средства для сохранения объектов векторных слоёв в GeoJSON-файлах. Полученный в результате оцифровки точечный слой сохраняем (экспортируем, выбирая СК, поля) в файл с расширением .geojson. В результате имеем пример нашего точечного слоя в формате GeoJSON.

Такой файл можно будет переслать и загрузить в виде векторного точечного слоя без каких-либо дополнительных действий в этом же или другом месте и проекте ГИС, или, наоборот, создание GeoJSON-файла можно запрограммировать для пересылки в любую ГИС, в том числе в QGIS .

```
{
  "type": "FeatureCollection",
  "bbox": [42.264429, 42.3678975, 57.4264367, 57.482453],
  "name": "depth_points_EPSG_4326",
  "crs": {
     "type": "name",
     "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" }
  },
  "features":
   [
     { "type": "Feature",
       "properties": { "depth": 2.0 },
       "geometry":  {
          "type": "Point",
          "coordinates": [ 42.266207, 57.446967 ]
       }
     },
     { "type": "Feature",
       "properties": { "depth": 2.0 },
       "geometry": {
          "type": "Point",
          "coordinates": [ 42.271742, 57.447930 ]
       }
     },
…
     { "type": "Feature",
       "properties": { "depth": 11.0 },
       "geometry": {
          "type": "Point",
          "coordinates": [ 42.366410, 57.456969 ]
       }
     },
     { "type": "Feature",
       "properties": { "depth": 11.0 },
       "geometry": {
          "type": "Point",
          "coordinates": [ 42.368532, 57.456597 ]
     } }
  ]
}
```


1.2 Особенности разработки в Web

Важная особенность Web - клиент-серверная архитектура [12]. Клиенты – браузеры: Chrome, Firefox, Microsoft Edge и другие; серверы: Apache 2, Nginx и другие. Передача данных между клиентами и серверами осуществляется с

использованием программ, называемых «протоколами», посредством клиентских HTTP-запросов и серверных HTTP-ответов. Для нашего приложения достаточно технологий, которыми обладают браузеры. В этом случае говорят о разработке «на стороне клиента» или «в среде клиента». Преимущество такого подхода – независимость от сервера и от каналов связи. Хотя при желании это приложение можно расположить в том числе и на сервере (о размещении приложения см. п. 1.2.4).

Далее будут упомянуты в том числе клиент-серверные приложения, в которых используются элементы процедур оцифровки, интересные с точки зрения нашей задачи.

1.2.1 Клиент-серверные Web-ГИС

Яркий представитель клиент-серверных ГИС – проект OpenStreetMap [30]. Это крупный многопользовательский проект и, соответственно, средства для оцифровки в рамках OpenStreetMap (OSM) направлены прежде всего на обеспечение многопользовательской работы. Оцифровка может осуществляться тремя редакторами:

- ID – это самый новый редактор доступный прямо на главном сайте во вкладке "правка". Он написан на HTML5/Javascript;
- Potlatch – это более старый основанный на Flash-редактор, который тоже доступен на вкладке "правка";
- JOSM - это мощный редактор (Java-приложение, требует установки Java версии 8 и более).

Понятно, что последние два редактора не подходят по определению, а редактор ID мог бы стать именно тем Web-приложением, которое нам необходимо. Но, к сожалению, этот редактор не подходит по крайней мере по следующей причине: нельзя использовать отдельно от OSM. Потребуется избыточно сложная переработка ID для оцифровки собственных карт на локальных участках.

К этому же типу проектов можно отнести NextGIS [29].

Другой тип клиент-серверных Web-ГИС – проект OpenLayers [31]. Этот проект мог бы использоваться для оцифровки, если бы имелся готовый редактор для нанесения объектов (точек в нашем случае). Однако в нём для нанесения объектов нужно писать JS-скрипты. Можно было бы, конечно, написать нужный нам редактор в рамках этого объекта. Однако это был бы либо клиент-серверный вариант, либо вариант с предварительной загрузкой достаточно объёмной js-библиотеки. Первый вариант отпадает по определению. Второй вариант представляется избыточно сложным для нашей относительно простой задачей.

Остальные клиент-серверные Web-ГИС и их инструменты оцифровки можно отнести к этим двум типам.

Резюмируя, можно заключить: имеющиеся библиотеки для клиент-серверных ГИС по сути являются надстройками для увеличения их функциональности. Невозможно непосредственно использовать эти библиотеки для нашей локальной задачи, а их доработка представляется избыточно сложной.

1.2.2 Клиентские Javascript-библиотеки с элементами ГИС

Здесь рассматриваются лишь открытые (OpenSourse) и бесплатные (Free Tools) JS-библиотеки, которые содержат готовые инструменты оцифровки или могли облегчить разработку Web-приложения для наших целей.

Отсеиваются библиотеки, которые подобно OpenLayers, упоминаемой выше, созданы для расширения функциональности OSM, Google Maps и других клиент-серверных сервисов, такие как: GeoCharts, Map Tools, Leaflet и подобные. Представляются также переусложнёнными библиотеки с зависимостями от других библиотек (jQuery + Raphael.js и т.п.), такие как: Mapael, Kartograph и подобные.

Среди JS-библиотек без зависимостей следует упомянуть, используемые

для создания интерактивных Web-карт [27,32,33]: 1) HighMaps (зависимость от jQuery, но хорошая документация); 2) amCharts 5: Maps; 3) anyMap; 4) D3.js.

Анализ показал, что функциональность этих библиотек позволяет работать с различными картографическими проекциями, что важно для больших земных участков, однако нет того, что необходимо для нашего случая – автоматизации создания точечного слоя с визуализацией по небольшой площади путём двумерной интерполяции. Представляется не совсем оправданным использование этих библиотек с избыточно богатой функциональностью для построения Web-приложения для нашей задачи.

1.2.3 Особенности разработки в среде клиента

Основные принципы создания Web-приложения для оцифровки продемонстрируем на примере использования библиотеки D3 и Google Maps API [34].

Вначале создаём HTML-документ и подключаем js-библиотеки. В тэге <body> помещаем div-контейнеры для размещения слоёв карты: второй предназначен для подложки – карты Google Map; первый – для слоя, создаваемого с использованием библиотеки D3. Сохраняем его в файле primer_D3+GoogleMapsAPI.html:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://d3js.org/d3.v4.min.js"></script>
    <script  src="https:
        //maps.googleapis.com/maps/api/js?key=YOUR_API_KEY">
    </script>
    <script src="./primer_D3+GoogleMapsAPI.js"></script>
</head>
<body>
    <div id="visualization"></div>
    <div id="googleMap"></div>
</body>
</html>
```

Для создания интерактивной карты с одним площадным и одним точечным объектом пишем JS-код (файл primer_D3+GoogleMapsAPI.js):

```
var data = [10,20,30,40,50];

var svg = d3.select("#visualization")
  .append("svg")
  .attr("width", 300)
  .attr("height", 200);

svg.selectAll("rect")
  .data(data)
  .enter()
  .append("rect")
  .attr("x", function(d, i) { return i * 50; })
  .attr("y", function(d) { return 200 - d; })
  .attr("width", 40)
  .attr("height", function(d) { return d; });

var center = { lat: 37.775, lng: -122.434 };
var map = new google.maps.Map(document.getElementById('googleMap'), {
  zoom: 12,
  center: center
});

var marker = new google.maps.Marker({
  position: center,
  map: map,
  title: 'Hello World!'
});
```

Как видно из JS-кода, для создания полигона с пятью вершинами в используется объект в формате svg, а в качестве карты – помещающися в фрагмент векторной Google-карты с центром с координатами: lat: 37.775, lng: -122.434.

Следует заметить, что с некоторого времени для использования Google Maps API необходимо получить платный ключ, который помещается вместо текста «YOUR_API_KEY « в скрипт в html-документе.

Таким образом, при разработке собственного Web-приложения необходимо создать div-элемент и в нём разместить карту-подложку и поверх этой карты создавать площадные и точечные объекты. Для размещения объектов JS обладает развитым механизмом обработки событий [11-14].

1.2.4 Платформы для размещения Web-приложения

Web-приложение в среде клиента, для разработки которого используется лишь «нативный» Javascript без каких-либо зависимостей от сторонних библиотек, обладает рядом преимуществ при выборе платформы для размещения. Такими платформами могут быть:

- браузер;
- web-сервер под Apache;
- web-сервер под Node.js;
- NW.js [28].

Запуск Web-приложения в браузере не должен вызывать вопросов. По сути – это «десктопный» вариант.

Для бесплатного использования на web-сервере также всегда можно найти соответствующего провайдера, поскольку для размещения приложения потребуется минимум ресурсов. Расширение возможностей приложения (например, использование СУБД) возможно двумя путями: под Apache, вероятнее всего придётся писать на PHP; под node.js – можно обойтись одним лишь JS.

Использование NW.js (ранее известного как node+webkit, webkit - популярный браузерный JS-движок), возможно требует небольшого пояснения, поскольку не так часто используется. Это также «десктопный» вариант, но при котором для JS доступны все ресурсы компьютера. Обеспечивается полная поддержка всех функций браузера, поэтому вариант, запускаемый в браузере, будет работать и под NW.js. Но, поскольку, становятся доступны ресурсы (в том числе системные) компьютера и JS-движка, то в этом случае JS по сути становится языком для разработки системных приложений и открываются соответствующие возможности для развития нашего приложения.

2 Проектирование Web-приложения

Особенность проектируемого приложения в том, что это клиентское приложение, использующее средства клиента-браузера, без использования средств сервера.

Рисунок 2.1 — Средства разработки Web-приложения

В качестве клиента можно использовать наиболее популярные браузеры: Google Chrome, Mozilla Firefox, Яндекс.Браузер, Microsoft Edge, Safari, поддерживающие стандарты Web, одобренные консорциумом W3C. В рамках этих стандартов:

- Javascript – используются средства ECMAScript 2015, которые поддерживаются во всех последующих версиях: ECMAScript 2016, ECMAScript 2017, ECMAScript 2018;
- HTML – используются средства HTML 5, но такие, которые будут работать и в HTML 4;

- CSS - используются средства CSS2 с некоторыми возможностями CSS3.

2.1 Основные элементы приложения

На рисунке 2.2 показана схема, на которой представлены основные блоки разрабатываемого Web-приложения.

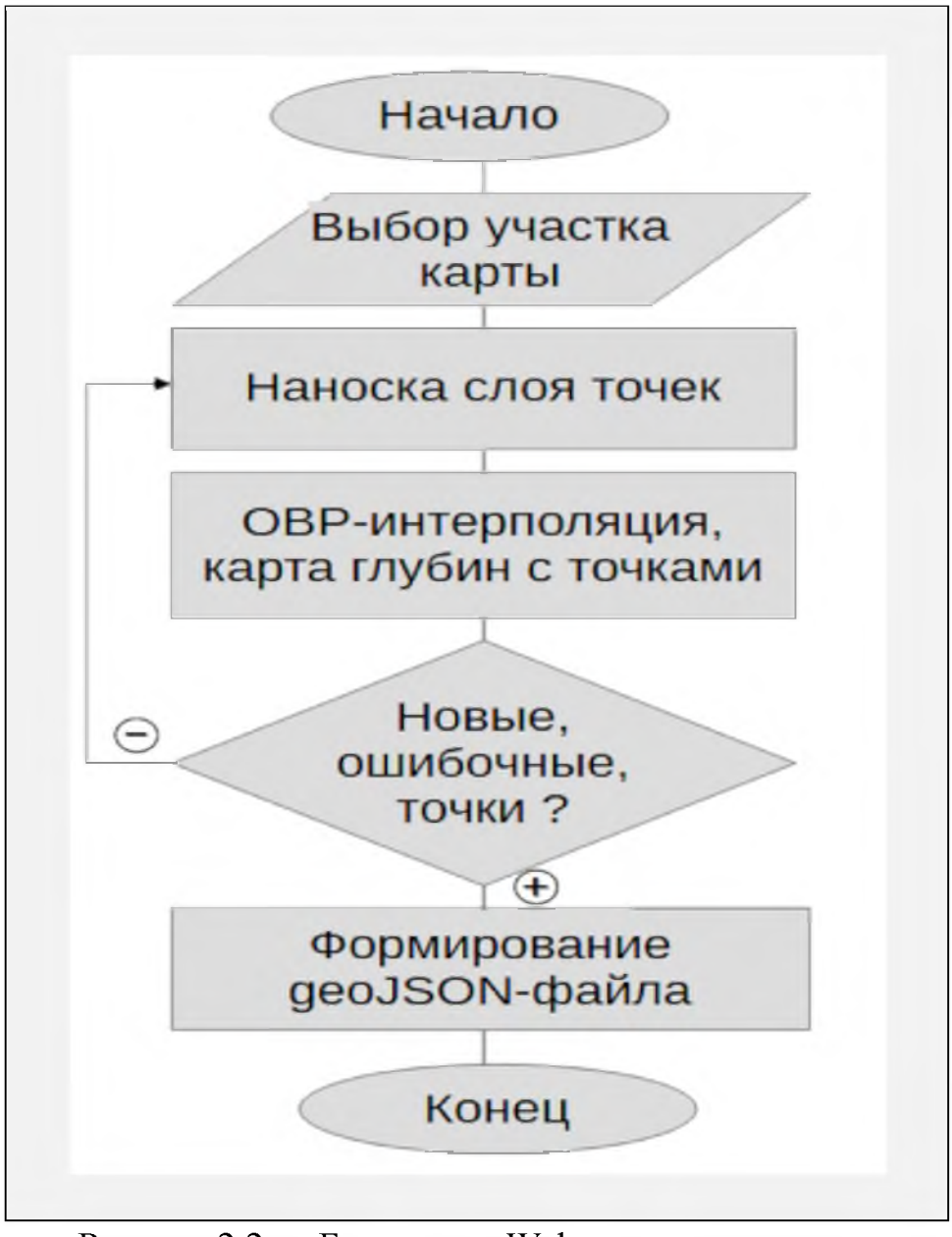

Рисунок 2.2 — Блок-схема Web-приложения

Для реализации приложения необходимо создать следующие основные структурные элементы:

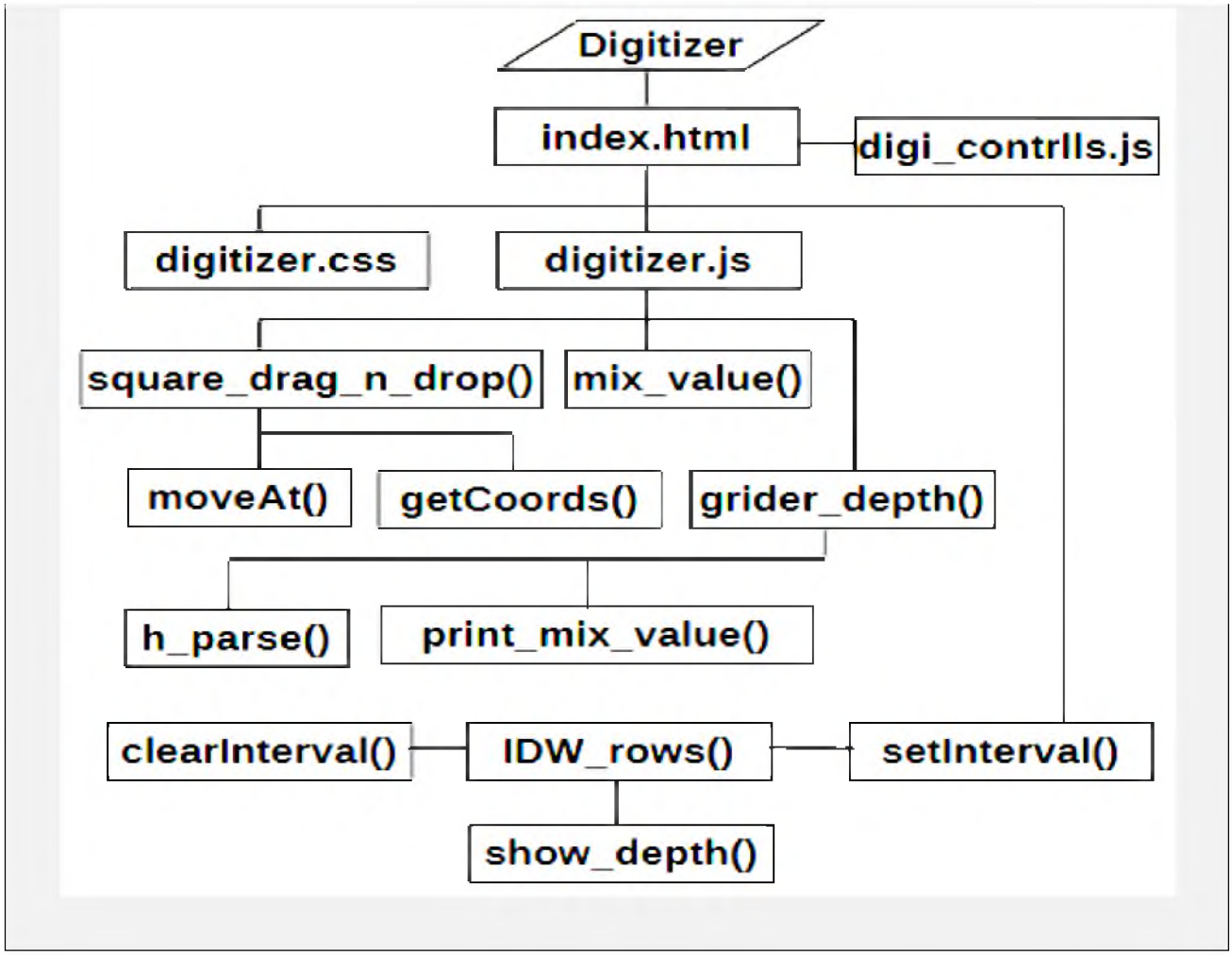

Рисунок 2.3 — Структурная схема Web-приложения

Кратко о возможностях структурных элементов:

•index.html – загружаются все html-элементы, CSS-стили и JS-функции с характеристиками по умолчанию с использованием анонимной функции, срабатывающей по событию окончания загрузки веб-страницы (window.onload=function(){}); на отображаемой html-странице размещаются все объекты (интерактивная схема-карта с выбором участка и средствами ввода точечных объектов, меню с событиями по запуску ОВР-интерполяции и записи точек с глубинами в JSON-файл;

•digi_controlls.js – содержит функции контроля вводимых значений;

•digitizer.js – головной модуль приложения, реализующий оцифровку вызовом функций на нижних уровнях;

•square_drag_n_drop() – функция для перемещения квадрата на схеме, выбора нужного участка и отображения его в виде карты с перевычислением и отображением координат (используя getCoords() и moveAt());

•mix_value() - при перемещении указателя мышки по карте выделяет клетку и показывает координаты верхнего левого угла клетки и глубину (м);

•grider_depth() – из GeoJSON-файла вводится бокс (координаты, ограничивающие участок карты) и массив точек с глубинами, если они имеются (с использованием h_parse()); строит привязанную к координатам клеточную область со слоем-фоном выбранного участка карты в качестве подложки; отображает на карте имеющиеся и новые точки с глубинами; пополняет JSON-файл вновь нанесёнными точками (с использованием h_parse()); отображает цветовую карту глубин (по результатам работы IDW_rows());

•IDW_rows() – ОВР-интерполяция глубин в клетки клеточной области. Интерполяция выполняется построчно путём вызова из таймера (setInterval()). Такой построчный способ позволяет интерполяции выполняться сколь угодно долго, обходя стандартную для браузеров защиту от длительно выполняющихся скриптов. По окончании интерполяции фильтруются артефакты в прибрежной зоне и расцвечиваются глубины в виде цветовой карты (с использованием show_depth()).

Далее приводятся фрагменты JS-кода основных функций Web-приложения.

2.1.1 Работа с картой

Загружается растровая карта-подложка в jpg-формате. Привязка карты осуществляется в базовой системе координат WGS 84 (Web Mercator, EPSG:3857). Координаты для привязки указываются в GeoJSON-файле. Используются географические координаты. Как показали эксперименты для небольших участков с размерами не более 10 км точность определения

координат, длин и площадей вполне достаточны (ошибка не превышает одного метра на один километр).

```
/*****************************************************/
/*  Загрузка значений по умолчанию                   */
/*****************************************************/
window.onload = function () {
    // определяем файл для привязки карты
    h_geoJSON = window["h_geoJSON_"+String(i_map)];
    // Вычисляем и отображаем координаты квадрата
 …
    /* Создаём объект с картой, соответствующей схеме */
    lat_top = h_geoJSON.bbox[3]; // широта верхней стороны карты
    lat_bottom = h_geoJSON.bbox[2]; // широта нижней стороны карты
    long_left = h_geoJSON.bbox[0]; // долгота левой стороны карты
    long_right = h_geoJSON.bbox[1]; // долгота правой стороны карты
    width_px_map = map_img[i_map].width; // Ширина в px.
    height_px_map = map_img[i_map].height; // Высота карты  в px.
    width_coords = long_right - long_left;
    height_coords = lat_top - lat_bottom;
    /*  Константы и переменные для выбранного квадрата */
    px_map = 800.0; // сторона квадрата  по умолчанию (px)
    delta_phi_square = px_map*(lat_top - lat_bottom)/height_px_map;
    min_vert_square =
       (delta_phi_square-Math.floor(delta_phi_square))*60;
    m_map =
       min_vert_square * m_mile; //длина (м) квадрата по вертикали
    m_1px = m_map / px_map; // метров/1пиксель
    px_1m = px_map / m_map; // пикселей/1метр
    delta_lambda_square =
        px_map*(long_right - long_left)/width_px_map;
    min_horiz_square =
       (delta_lambda_square-Math.floor(delta_lambda_square))*60;
    // Рассчитываем длину стороны квадрата на схеме в px
    length_square_schema =
        schema_map_img[i_map].height/map_img[i_map].height*px_map;
…
    //  Вычисляем некоторые параметры квадрата по умолчанию.
    //      При передвижении квадрата - перевычисляются.
    delta_lambda = long_right_squre_decy - long_left_squre_decy;
    delta_phi = lat_top_squre_decy - lat_bottom_squre_decy;
    step_lambda = delta_lambda/n;
    step_phi = delta_phi/n;
} // End window.onload
```

На карте с использованием кнопки мышки выбирается участок в виде квадрата для оцифровки (используется технология drap-and-drop).

```
/*****************************************************************/
/*Захватить-протянуть-отпустить квадрат на схеме (drag-and-drop)*/
/*  Перевычисляются координаты на экране в координаты карты     */
/*****************************************************************/
function square_drag_n_drop(event) {
  e = event || window.event;
  var square = document.getElementById("square_schema");
  var schema = document.getElementById("back_img");
  var rect_square = square.getBoundingClientRect();
  var rect_schema = schema.getBoundingClientRect();
  //var computedstyle_schema = getComputedStyle(schema);
  var computedstyle_schema = getComputedStyle(div_schema_back_img);
  var computedstyle_square = getComputedStyle(div_square);
  var coords = getCoords(square);
  shiftX = e.pageX - coords.left; shiftY = e.pageY - coords.top;
  square.style.position = 'absolute'; moveAt(e);
  function moveAt(e) {
    square.style.left = e.pageX - shiftX + 'px';
    square.style.top = e.pageY - shiftY + 'px';
    var rect_square = square.getBoundingClientRect();
    var rect_schema = schema.getBoundingClientRect();
    /*Вычисляем координаты углов выбранного квадрата
     и отображаем их во входных данных как град.мин.сек.*/
    start_y_px_schema = rect_square.top-rect_schema.top;
    lat_top_squre_decy =
       lat_top - start_y_px_schema*height_coords/height_px_schema;
    var rab_min =
       (lat_top_squre_decy - Math.floor(lat_top_squre_decy))*60;
…
  } //End moveAt(e)
  document.onmousemove = function(e) { moveAt(e);};
  square.onmouseup = function() {
    document.onmousemove = null;
    square.onmouseup = null;
  };
  square.ondragstart = function() {
    return false;
  };
  function getCoords(elem2) { // кроме IE8
      var square = elem2.getBoundingClientRect();
      var computedstyle_square = getComputedStyle(elem2);
      return {
          top: Number(square.top) + pageYOffset -
Number(computedstyle_square.borderTopWidth.substr(0,1)),
          left: Number(square.left) + pageXOffset -
Number(computedstyle_square.borderLeftWidth.substr(0,1))
      };
  }
} // End square_drag_n_drop()
```

При проведении указателя мышки по квадрату карты выводятся координаты и глубина (м) пикселя (размеры пикселей - вычисляемые

величины).

```
/*********************************************************/
/* Функция mix_value()  При наведении мышкой показывает */
/* координаты клетки и глубину (м)                       */
/*********************************************************/
function mix_value(event,elem) {
    var z_cell;
    var z_il_cell;
    var h_cell;
    var dist_cell;
    var flows = new Array(9);
    // Получить ограничивающий квадрат для участка карты.
    var br=elem.getBoundingClientRect();
    event = event || window.event
    var body = document.body;
    var docElem = document.documentElement;
    var l_td=px_map/n; // размер стороны клетки в пикселах
    // Вычислить прокрутку документа.
    var scrollTop = window.pageYOffset || docElem.scrollTop ||
body.scrollTop
    var scrollLeft = window.pageXOffset || docElem.scrollLeft ||
body.scrollLeft
    // Документ(html или body) бывает сдвинут относительно окна (IE)
    var clientTop = docElem.clientTop || body.clientTop || 0
      var clientLeft = docElem.clientLeft || body.clientLeft || 0
    // Прибавляем относ-но окна прокрутку и вычитаем сдвиг html/body
    e_x=(event.pageX-(br.left+1+scrollLeft-clientLeft))/l_td;
    e_y=(event.pageY-(br.top+1+scrollTop-clientTop))/l_td;
    e_x=Math.round(e_x);
    e_y=Math.round(e_y);
    if(typeof h == "undefined" || typeof h[e_y] == "undefined" ||
typeof h[e_y][e_x] == "undefined") {h_cell=""}
      else {h_cell=Math.round(h[e_y][e_x]*100)/100};
    document.getElementById("y_out").value=e_y;
    document.getElementById("x_out").value=e_x;
    document.getElementById("h_in_cell").value=h_cell;
    // Вычисляем и отображаем координаты клетки
    lat_point = lat_top_squre_decy - e_y*delta_phi_square/n;
    var rab_min_start = (lat_point - Math.floor(lat_point))*60;
    var rab_sec_start = ((lat_point -
Math.floor(lat_point))*60-Math.floor(rab_min_start))*60;
    var lat_point_text_input =
String(Math.floor(lat_point))+"&#176;" +
String(Math.floor(rab_min_start))+"&prime;" +
String(Math.round(rab_sec_start*100)/100)+"&Prime;";
    document.getElementById("phi_g_mob").innerHTML =
lat_point_text_input;
    document.getElementById("phi_d_mob").innerHTML =
String(Math.floor(lat_point*100000000)/100000000)+"&#176;";
    long_point = long_left_squre_decy + e_x*delta_lambda_square/n;
…
    document.getElementById("lambda_d_mob").innerHTML =
String(Math.floor(long_point*100000000)/100000000)+"&#176;";
```

Работа блока элементов работы с картами проверялась на примере карты Новороссийской (Цемесской) бухты.

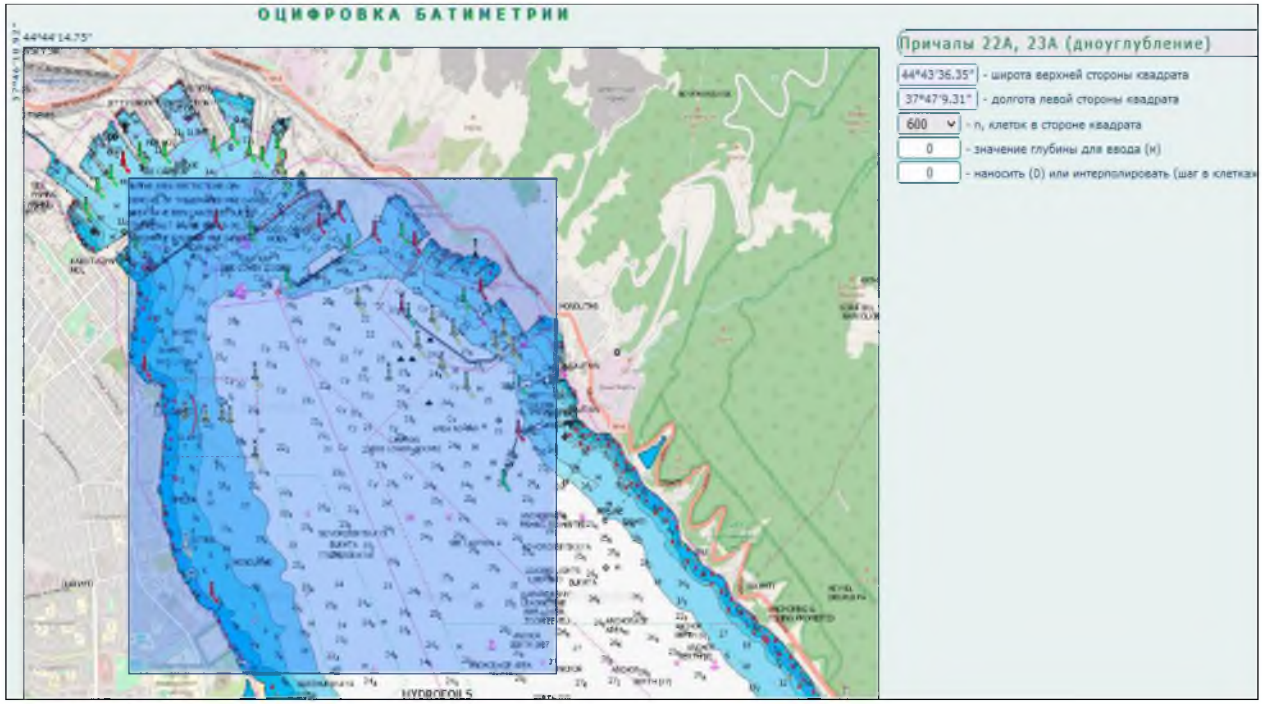

Рисунок 2.4 — Выбор участка для оцифровки на карте Цемесской бухты

На рисунке показан фрагмент навигационной карты (проекция Mercator/WGS84), совмещённый с Яндекс-картой (проекция Web Mercator). Рисунок демонстрирует хорошее совмещение небольших участков карт, несмотря на различия проекций.

2.1.2 Методы оцифровки

Собственно оцифровка, нанесение точечного слоя с глубинами на фоне обновляемой карты глубин и сохранение полученных данных о точках в JSON-файле, выполняется с помощью функции grider_deph(), в которой создаётся клеточная область с оцифровываемым фрагментом карты, вызываются функции h_parse()и print_mix_value() и создаётся цветовая карта глубин по данным обновляемого глобального массива глубин в клетках области по данным функции IDW_depth().

```
/*************************************************************/
/*Функция grider_depth():                                    */
/*построение таблицы-сетки с фоном карты-подложки;           */
/*ввод-вывод массива батиметрии из geoJSON-файла;            */
/*построение цвет.карты по глоб. массиву глубин из IDW_depth()*/
/*расчёт площади зеркала и объёма воды для текущей батиметрии */
/*************************************************************/
function grider_depth() {
    step_lambda = delta_lambda/n; step_phi = delta_phi/n;
    m_1cell = m_map/n; // длина стороны клетки (метров/1клетку)
    m2_1cell=m_1cell*m_1cell; //площадь клетки (кв.метров/1клетку)
    cell_1m = px_1m * (n / px_map); // клеток/1метр
    var lat_top_squre_px=-(lat_top-lat_top_squre_decy)*
height_px_map/(lat_top-lat_bottom);
    var long_top_squre_px=-(long_left-long_left_squre_decy)*
width_px_map/(long_left-long_right);
    document.getElementById("rez_1").style.marginTop =
 - (px_map+70)+"px";
    document.getElementById("contaner_3").style.width =
 px_map+15+"px";
    document.getElementById("contaner_3").style.height =
 px_map+15+"px";
    // Показываем прогресс-бар, формируем клет-ю область карты
    progress_bar.style.display="block";
    var tbl=
"<table       id='depths'       onMouseOver='mix_value(event,this)'
      onClick='print_mix_value(event,this)'
onContextMenu='print_br()' style='background-position: "+
       Math.round(long_top_squre_px+1)+"px
"+Math.round(lat_top_squre_px+1)+"px;            background-image:
URL("+map_img[i_map].src+");'>";
    // Определяем размер клетки
    var h_dt = px_map/n+"px";  var w_dt = h_dt;
    for (var i_str=0; i_str<n; i_str++) {
        tbl+="<tr id=i_str>";
        for (var i_col=0; i_col<n; i_col++) {tbl+=
"<td id=i_col style='height:"+h_dt+";width:"+w_dt+"'></td>";
        } tbl+="</tr>";
    } tbl+="</table>";
    document.getElementById("grid").innerHTML=tbl;
…
    // Формируем и показываем шкалу глубин
…
    // Вводим батиметрию из geoJSON-файла + дополняем новыми данными
    h_bath.length = 0;
    h_bath=h_parse(long_left_squre_decy,       lat_top_squre_decy,
long_right_squre_decy, lat_bottom_squre_decy);
…
} // End function grider_depth()
```

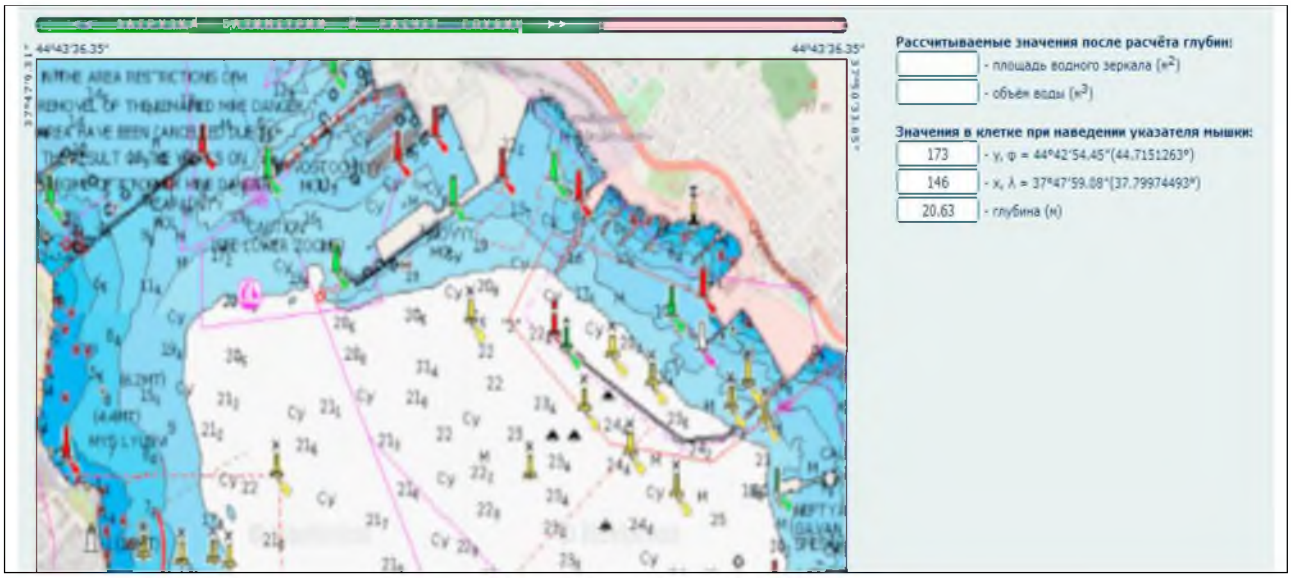

Рисунок 2.5 — Фрагмент участка Цемесской бухты, выбранный для оцифровки

2.1.3 Интерполяция методом ОВР

Итак, исходя из задачи, решаемой в настоящей работе, достаточным представляется метод обратных взвешенных расстояний (ОВР, или от англ. inverse distance weighting – IDW). При этом подразумевается, что в рассматриваемом варианте ОВР-алгоритма используется интерполяция глубин (без сглаживания) по данным в нерегулярно расположенных точках на середину клеток клеточной области с сохранением известных значений «как есть» при совпадении местоположения точки и клетки. К числу существенных для нашей задачи достоинств ОВР-алгоритма следует отметить простоту реализации и то, что алгоритм не «зависает» из-за мультиколлинеарности данных [4].

Используя аппарат обозначений Кнута для оценки трудоемкости алгоритмов, среди недостатков ОВР-алгоритма отмечают [3] существенный для решаемой здесь задачи – трудоемкость интерполяции (O(N)) на больших объемах исходных данных по сравнению, например, с модифицированным методом Шепарда, обладающим умеренной трудоемкостью (O(N·logN)) – время построения модели, O(logN) – время интерполяции). Но предварительные эксперименты дают основание полагать, что достаточным является локальный вариант ОВР-алгоритма, когда для интерполяции используется небольшое

число (около 6-ти) ближайших точек. Кроме этого, область поиска влияющих точек осуществляется в пределах квадрата, покрывающего не весь район, а лишь клеточную область.

Поскольку квадрат выбирается много раз в пределах одного и того же района, то выборку точек, покрываемых квадратом, а затем и ближайших точек для интерполяции можно ускорить, предварительно упорядочив все точки оцифрованных глубин района по широте и долготе. Тогда, затратив один раз время на предобработку, трудоемкость (по времени) с учетом многократного использования алгоритма интерполяции можно существенно уменьшить до O(logN).

Особенности предлагаемой реализации ОВР метода можно рассмотреть на фрагменте кода:

```
/*************************************************************/
/*Функция интерполяции в узлы регулярной сетки методом ОВР    */
/* с отбором ближайших соседей линейным методом               */
/*по строкам. По окончании  интерполяции запускается show_depth()
фильтрует одиночные артефакты (на береговой линии) и расцвечивает
глубины                                                       */
/*IDW_rows()вызывается по Click в index.htm                   */
/*************************************************************/
function IDW_rows() {
    // h_bath - исходный глобальный массив точек с глубинами
    // в GEOJson-формате (h_bathy.length*3). Триплет: long,lat,h
    // n - число узлов сетки (n*n) для интерполяции глубин
    var d_h = 0;
    var n_min = 6; //Число отбираемых ближайших точек
    var d_min=new Array(n_min);//Массив расстояний до этих точек
    var kk_min = new Array(n_min);//Массив индексов этих точек
        // в geoJSON-массиве
        var lat_i=lat_top_squre_decy-step_phi*i;//шир. клетки
        for (var j=0; j<n; j++) {
            var long_j=long_left_squre_decy+step_lambda*j;//долг.
            var w; var sum_w = 0; var sum_wz = 0;
            /* Отбираем n_min ближайших точек для интерполяции*/
            /*  проверяем все точки исходных значений,        */
            /* то есть используем метод прямого перебора.     */
       metka_2: for (var k_min=0; k_min<n_min; k_min++) {
                d_min[k_min]=100000000;
          metka_1: for (var kk=0; kk<h_bath.length; kk++) {
                   var phi_point = h_bath[kk][1] * Math.PI/180;
                   var phi_cell = lat_i * Math.PI/180;
                   var d_phi = phi_point - phi_cell;
                   var lambda_point = h_bath[kk][0] * Math.PI/180;
                   var lambda_cell = long_j * Math.PI/180;
```

```
                var d_lambda = lambda_point - lambda_cell;
                var d_rab_1=2*
            Math.asin(Math.sqrt(Math.pow(Math.sin(d_phi/2),2)+
              Math.cos(phi_cell)*
              Math.cos(phi_point)*
              Math.pow(Math.sin(d_lambda/2),2)))*
              km_r_spher;//в км
                var d_rab = Math.pow(d_rab_1,2);
                // Игнорируем точки с батиметрией, располо-
                //женные далее 1 стороны квадрата
                //от текущей клетки
                if (d_rab_1 * 1000 > 1*m_map)  {
                    continue metka_1;
                }
                //Интерполируем только лишь, если нет
                //совпадения интерполируемой точки с точкой
                //из исходного массива
                if (d_rab_1 > 0) {
                    if(k_min == 0) {
                        if(d_rab<d_min[k_min]) {
                            d_min[k_min]=d_rab;
                            kk_min[k_min]=kk;
                        }
                    }
                    if (k_min > 0) {
                        if(d_rab<d_min[k_min]) {
                            //Игнорируем точку, если она
                            //совпадает с любой предыдущей
                            for (var kkk=1;kkk<=k_min;kkk++) {
                                if ((h_bath[kk][0] ==
                                  h_bath[kk_min[kkk-1]][0])
                                   &&(h_bath[kk][1] ==
                                  h_bath[kk_min[kkk-1]][1]))
                                {
                                    continue metka_1;
                                }
                            }
                            d_min[k_min]=d_rab;
                            kk_min[k_min]=kk;
                        }
                    }
                }
                    else { //Как только точка ближе одной
                    //клетки, то для клетки берём глубину
                    //в этой точке
                        h[i][j]=h_bath[kk][2];
                        break metka_2;
                }
              } // отобрали очередную ближайшую точку
              /*Рассчитываем веса, взвешиваем глубины,
                 накапливаем суммы */
             if (h[i][j]==-1) {
                 w=1/d_min[k_min];
```

```
                    sum_w=sum_w+w;
                    sum_wz=sum_wz+w*h_bath[kk_min[k_min]][2];
                }
            }
            // рассчитываем значение глубины
            if (h[i][j] == -1) { //глубина ещё не рассчитывалась
                                 if(sum_w > 0) {
                                       h[i][j]=sum_wz/sum_w;
                                 }
                h[i][j]=h[i][j]+d_h*h[i][j]*
                      Math.random()-d_h*h[i][j]/2;
            }
            if (h[i][j] <= coast) { // это суша или берег.линия
                h[i][j] = 0;
            }
        document.getElementById("into_progress_bar").style.width=
             (i+1)* px_map/n+"px";
        }
        // Заполнили очередную строку, продвигаем прогресс-бар
        into_progress_bar.style.width=
             ((i+1)*px_map/n)-1+"px";
    i++; // переходим к следующей строке
    if (i == n) { // если закончили в последней строке
        // Скрываем прогресс-бар
        progress_bar.style.display="none";
        clearInterval(end_IDW);
        //Отображаем глубины с фильтрацией одиночных артефактов
        // и рассчитываем площадь водного зеркала и объём воды
        alfa=1; // 1 - +расцвечиваем глубины
             show_depth();
    };
} // End IDW_rows()
```

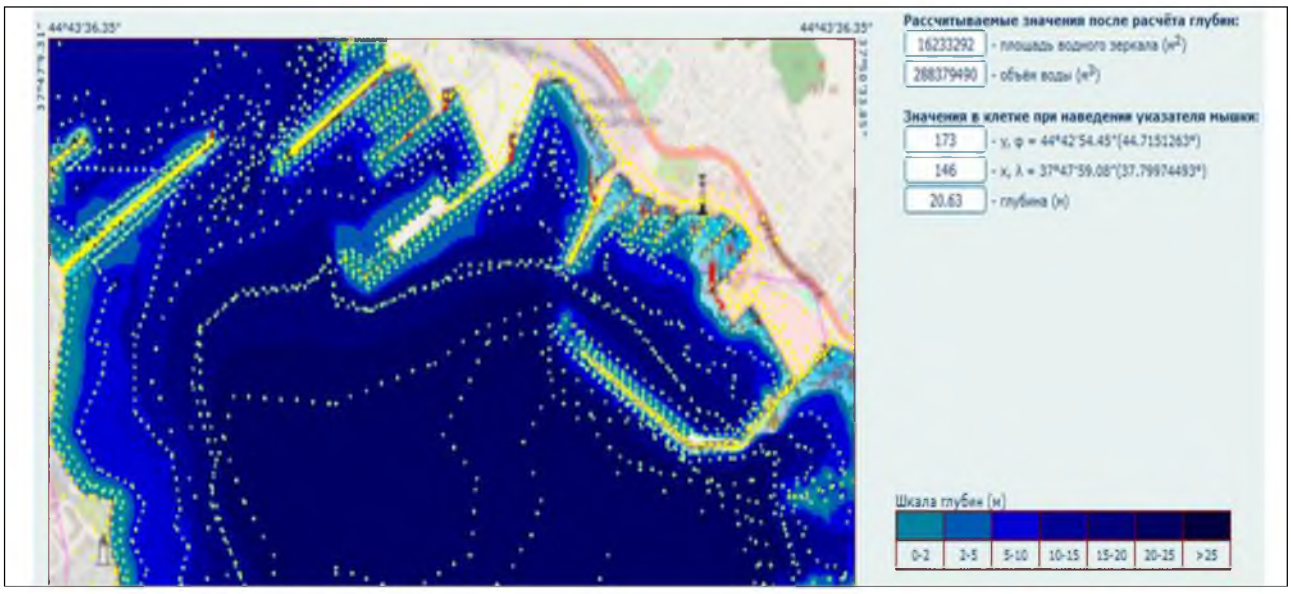

Рисунок 2.6 — Фрагмент участка Цемесской бухты после оцифровки

2.1.4 GeoJSON-формат для сохранения и экспорта результатов

Файл bathy.js в формате GeoJSON используется в grider_depth() для ввода исходных данных и вывода результатов в процессе оцифровки. По окончании оцифровки результаты сохраняются в этом файле, который может быть использован для передачи информации о глубинах в другие программы, поддерживающие GeoJSON.


```
 {
"type": "FeatureCollection",
"crs": {
    "type": "name",
    "properties": {
        "name": "urn:ogc:def:crs:EPSG::3857"
    }
},
"bbox": [37.771922, 37.885556, 44.656613, 44.737430],
"features": [
    {
        "type": "Feature",
        "geometry": {
            "type": "Multipoint",
            "coordinates": [
[37.780287176658, 44.671643922686, -5.00],

[37.788704509991, 44.677224381379, -5.00],

[37.788143354436, 44.685993673611, -5.00],
[37.777481398880, 44.733826176694, -5.00],

[37.861654732214, 44.709112716768, -5.00],
[37.871194376658, 44.701937841305, -5.00],

[37.773553309991, 44.663273234647, -1.00],
[37.787582198880, 44.666860672378, -1.00],
```

```
...
[37.783048715521, 44.725994992761, 5.00],
[37.780207865521, 44.727609314858, 5.00],
[37.782605049289, 44.726247107516, 5.000],
[37.782161383058, 44.726499222270, 5.000],
[37.781717716826, 44.726751337025, 5.000],
[37.781274050595, 44.727003451780, 5.000],
[37.780830384363, 44.727255566535, 5.000],
[37.780386718132, 44.727507681290, 5.000],
[37.781060120521, 44.728012895382, 5.00],
[37.780680611536, 44.727833181054, 5.000],
[37.782622588021, 44.727003944072, 5.00],
[37.781480532272, 44.727741417744, 5.000],
[37.781900944024, 44.727469940105, 5.000],
[37.782321355775, 44.727198462466, 5.000],
[37.783616885521, 44.727407524596, 5.00],
...
[37.856656488889, 44.659106394066, 20.00],
[37.857778800000, 44.658707789873, 20.00],
[37.858339955556, 44.658309185681, 20.00],
[37.858901111111, 44.657910581489, 20.00],
[37.859462266667, 44.657511977296, 20.00]
]
}
}
]
}
```

2.2 Пример использования

Использование разработанного web-приложения покажем на примере участка Горьковского водохранилища, ранее оцифрованного в QGIS, чтобы можно было сравнить результаты, полученные разными способами.

2.2.1 Привязка карты

Выбираем карту и вырезаем необходимый участок на карте или, выражаясь терминологией ГИС, загружаем, привязываем и обрезаем растр, готовим карту-подложку.

При этом помним, что в нашем приложении используется CRS (coordinate reference system) Web-Mercator (EPSG:3857).

Рисунок 2.7 — Выбираем необходимый участок карты

2.2.2 Оцифровка

Как уже упоминалось для оцифровки предусмотрены два инструмента: наноска одиночных точек и наноска группы точек с указанным шагом между предварительно нанесёнными одиночными точками.

Последний инструмент существенно упрощает и уточняет оцифровку глубин по сравнению с тем, как это приходится делать в QGIS.

Кроме этого, дополнительные преимущества можно получить за счёт свойства многооконности браузеров, когда одно и то же веб-приложение можно запускать в разных окнах.

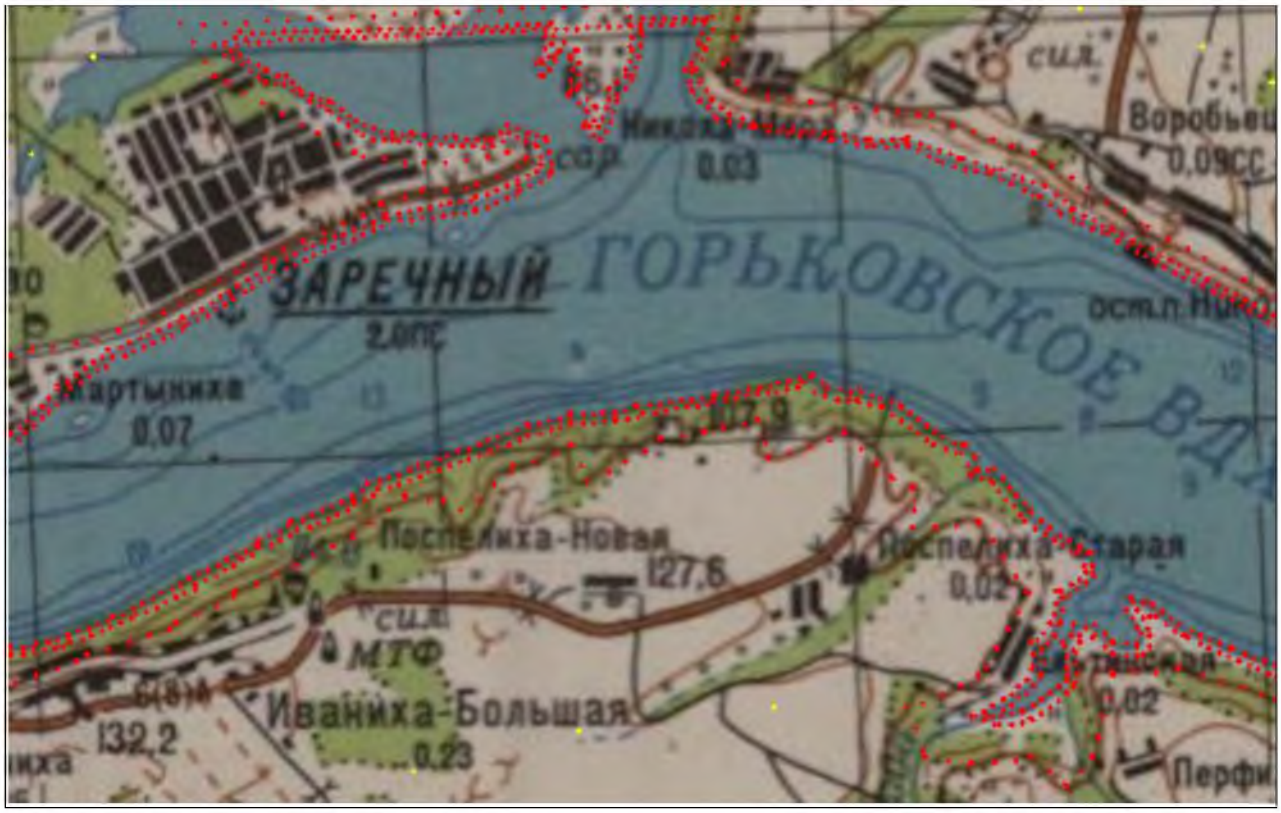

Рисунок 2.8 — Первое окно в браузере с точками береговой линии и бровки

Рисунок 2.9 — Второе окно в браузере с точками береговой линии и бровки после построения полигона ОВР-интерполяцией

Рисунок 2.10 — Третье окно в браузере со всеми точками оцифровки

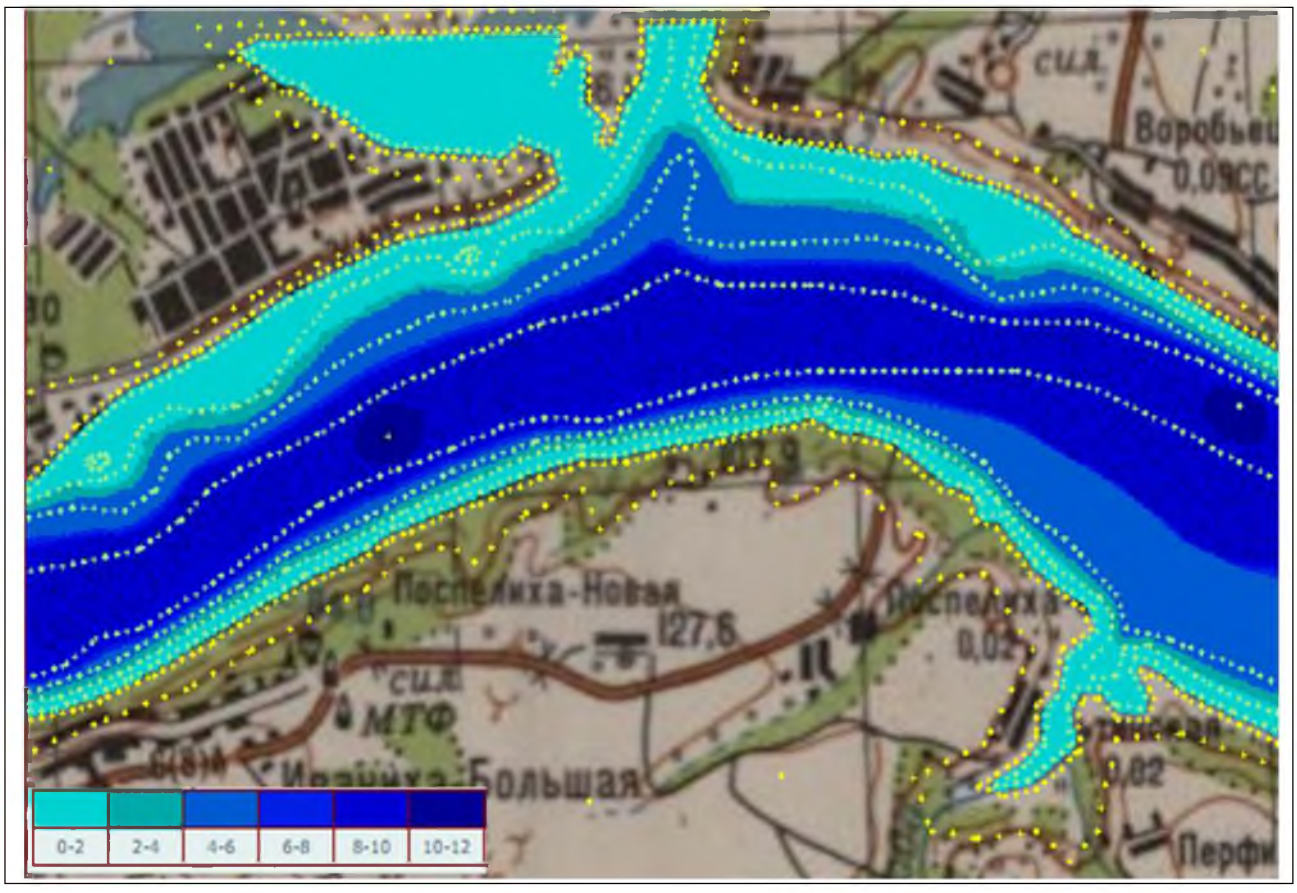

Рисунок 2.11 — Четвёртое окно в браузере со всеми точками оцифровки после ОВР-интерполяции

2.2.3 Интерполяция методом OBP

Выше, при оцифровке, уже были показаны примеры применения OBP-интерполяции, как вспомогательное средство помогающее визуально просматривать и корректировать процедуру наноски точек. Однако OBP можно использовать и для анализа водного объекта по результатам оцифровки. Для этого отобразим цветовую батиметрическую карту с более подробной цветовой шкалой глубин.

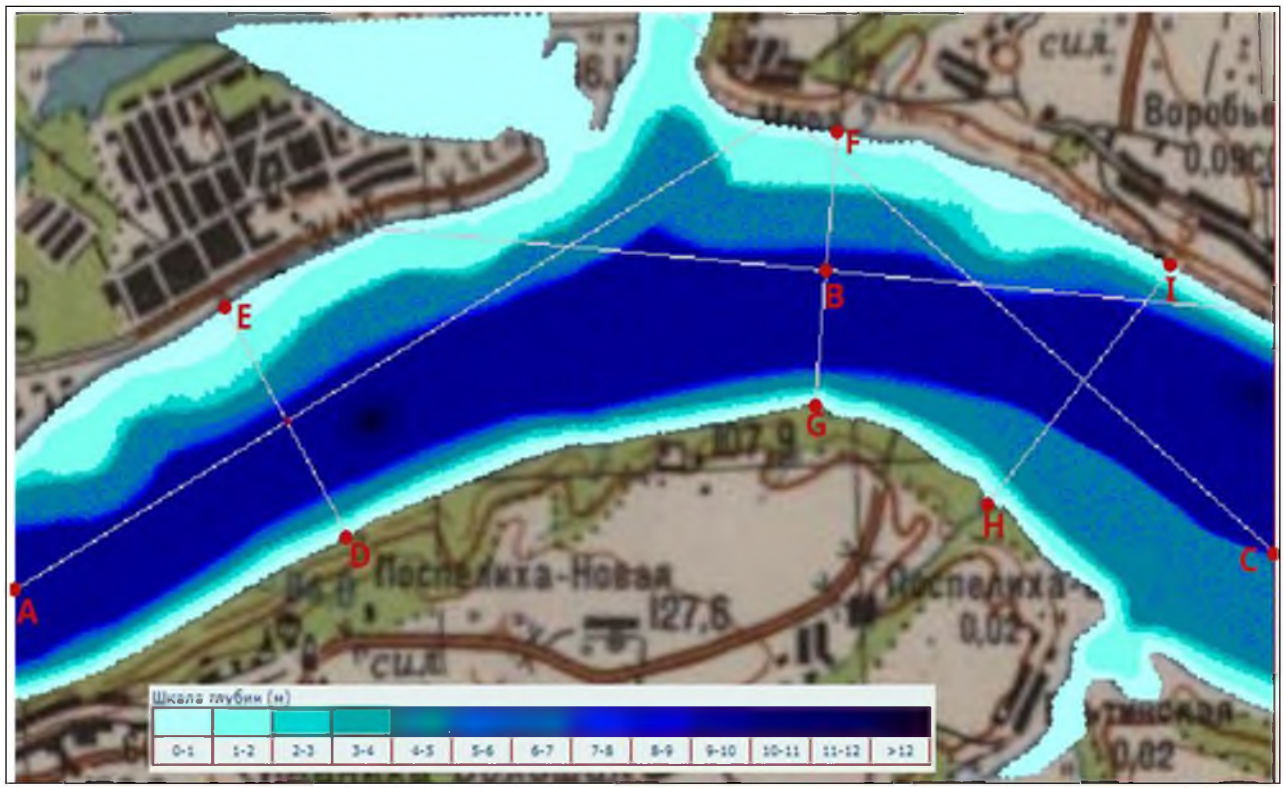

Рисунок 2.12 —OBP-интерполяция результатов оцифровки в веб-приложении (показано также расположение точек и створов для промеров)

Можно сравнить эту карту с картой, построенной в QGIS (см. рисунки 1.6 и 1.7). Нетрудно заметить преимущество последней карты из-за отсутствия артефактов, имеющихся на карте QGIS. Гладкость рельефа обеспечивается за счёт более равномерного расположением точек автоматизацией оцифровки, а также OBP-интерполяцией в клеточную область с регулируемым размером клеток вместо растра с узлами и методом выбора окрестности расположения влияющих точек, обеспечивающим интерполяцию вместо экстраполяции.

В веб-приложении также реализованы инструменты для расчёта длин, площадей и объёмов, которые можно использовать для определения морфометрических характеристик водного объекта, подобных полученным ранее в QGIS.

Значения характеристик, полученные в Web-приложении и в QGIS, и расхождения между ними приведены в таблице.

Таблица 2.1 — Сравнение характеристик участка Горьковского водохранилища

Характеристика	QGIS	Web-приложение	Разница (QGIS –Web)	Разница (|% к Web|)
Длина по ABC, м	6515	6528	13	0,20
Ширина по DE, м	1178	1174	-4	0,34
Ширина по FG, м	1233	1235	2	0,16
Ширина по HI, м	1381	1376	-5	0,36
Площадь зеркала, км^2	9,1547	9,3572	0,2025	2,16
Объём воды, км^3	0,0407002	0,052016	0,011316	21,75
Максимальная глубина, м	12,548	12,995	0,447	3,44
Средняя глубина, м	4,422	5,560	1,138	20,47
Размеры пикселя, м	17,6475x26,8201	9,9992x9,9992	-	-
Пикселей/клеток	19342	93587	-	-

Для сравнения длин использовались точки с координатами:

Таблица 2.2 — Координаты точек для сравнения длин (CRS Web Mercator) *)

Точка	Широта	Долгота
A	57°27′04.42″E (57.45122711°)	42°15′55.94″E (42.26553799°)
B	57°47′27.83″E (57.46328523°)	42°19′48.24″E (42.33006696°)
C	57°27′12.84″E (57.45356674°)	42°21′57.03″E (42.36584208°)
D	57°27′09.60″E (57.45266688°)	42°17′31.63″E (42.29211857°)
E	57°27′42.64″E (57.46184546°)	42°16′56.72″E (42.28242251°)
F	57°28′07.91″E (57.46886437°)	42°19′51.85″E (42.33107000°)
G	57°27′28.39″E (57.45788607°)	42°19′45.23″E (42.32923109°)
H	57°27′14.14″E (57.45392669°)	42°20′34.58″E (42.34293931°)
I	57°27′48.15″E (57.46337522°)	42°21′28.14″E (42.35781776°)
*) Координаты определены в Web-приложении, в QGIS брались ближайшие		

Как следует из таблицы 2.1 невязки для длин можно считать несущественными и можно сделать вывод, что длины, а отсюда и площади в

Web-приложении рассчитываются с приемлемой точностью, несущественно отличающейся от точности QGIS. Этот вывод касается картографических свойств Web-приложения.

Для более уверенного заключения необходимы проверки для различных широт на картах с линейными и площадными объектами, размеры которых известны.

Точность вычисления глубин и объёмов связана со свойствами конкретной реализации ОВД-интерполяции, поэтому не представляется возможным отдать предпочтение какой-нибудь реализации, хотя визуально более приемлем гладкий и без артефактов рельеф Web-приложения.

Возможно, что увеличение глубин и, соответственно, объёма связано с более точной оцифровкой и существенным увеличением количества точек, а также с уменьшением количество выпуклостей в виде артефактов ОВД-интерполяции в Web-приложении.

Представляется, что для заключения по качеству ОВР следует провести подобное сравнение с программой, в которой способы оцифровки подобны реализованным в Web-приложении, например с ОВР в пакете Surfer [5]. К этому следует добавить возможность тестирования с использованием метода «кросс-валидации» в контрольных точках, когда ОВД-значения в этих точках сравниваются с измеренным значением без использования этого значения в процедуре интерполяции.

2.2.4 Результаты в формате JSON

Ниже представлен фрагмент результатов оцифровки Горьковского водохранилища в формате GeoJSON.

Забегая вперёд следует отметить, что файл с этими результатами без ошибок воспринимается QGIS и отображается в виде соответствующего точечного слоя, а двумерная ОВД-интерполяция приводит к приемлемым результатам, похожим представленным на рисунках 1.5, 1.6, 2.12.

```
{
    "type": "FeatureCollection",
    "crs": {
        "type": "name",
        "properties": {
            "name": "urn:ogc:def:crs:EPSG::3857"
        }
    },
    "bbox": [42.263832, 42.3678975, 57.4264367, 57.482453],
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Multipoint",
                "coordinates": [
[42.264525770000, 57.444641997500, -3.00],
[42.270422815000, 57.445855684000, -3.00],
[42.266145812854, 57.444975422824, -3.000],
[42.267765855707, 57.445308848148, -3.000],
[42.269385898561, 57.445642273473, -3.000],
[42.277707400000, 57.447256091500, -3.00],
[42.272056195715, 57.446169689342, -3.000],
[42.273689576430, 57.446483694684, -3.000],
[42.275322957145, 57.446797700026, -3.000],
[42.276956337860, 57.447111705368, -3.000],
[42.282563790000, 57.448656499000, -3.00],
…
[42.320027370000, 57.457712467500, 2.00],
[42.314293155391, 57.457225108219, 2.000],
[42.315149755781, 57.457297911938, 2.000],
[42.316006356172, 57.457370715658, 2.000],
[42.316862956562, 57.457443519377, 2.000],
[42.317719556953, 57.457516323096, 2.000],
[42.318576157343, 57.457589126815, 2.000],
[42.319432757734, 57.457661930534, 2.000],
[42.326618185000, 57.458272630500, 2.00],
…
[42.367724057500, 57.455471815500, 10.00],
[42.353425899493, 57.458921926681, 10.000],
[42.354217238985, 57.458730978363, 10.000],
[42.355008578478, 57.458540030044, 10.000],
[42.355799917970, 57.458349081725, 10.000],
[42.356591257463, 57.458158133407, 10.000],
[42.357382596956, 57.457967185088, 10.000],
[42.358173936448, 57.457776236769, 10.000],
[42.358965275941, 57.457585288451, 10.000],
[42.359756615433, 57.457394340132, 10.000]
]
}
}
]
}
```

Указаны свойства геоданных, в том числе:

- EPSG-код нашей CRS Web-Mercator в реестре проекций - "urn:ogc:def:crs:EPSG::3857";
- координаты углов участка карты с частью Горьковского водохранилища (бокс) - [42.263832,42.3678975,57.4264367,57.482453];
- тип геометрии слоя - "Multipoint" и массив точек с координатами и глубинами.

Заключение

В соответствии с заданием в работе были рассмотрены возможности использования Web-технологий для оцифровки глубин водных объектов по растровым картам, подобно тому, как это возможно с использованием ГИС QGIS. В то же время разрабатываемое Web-приложение необходимо для оцифровки небольших акваторий, когда применение сложных программ типа ГИС нецелесообразно.

Анализ работы в QGIS показал, что процесс оцифровки, включающий этапы привязки карты к системе координат Web Mercator и наноски точечного слоя с глубинами, должен быть дополнен ОВР-интерполяцией по площади для визуального анализа рельефа дна. Кроме этого, необходимо уменьшить трудоёмкость ручной работы за счёт автоматизации оцифровке изобат. Необходима модификация метода ОВР для интерполяции не в узлы растра, а в клеточную область с задаваемыми размерами клетки. Окрестность расположения влияющих точек, должна обеспечивать интерполяцию вместо экстраполяции.

После рассмотрения особенностей Web-технологий выбрано проектное решение по разработке Web-приложения в среде клиента (браузера) без использования средств сервера с программированием на языке Javascript (с HTML и CSS).

Разработанное Web-приложение апробировано на примере участка Горьковского водохранилища, ранее оцифрованного средствами QGIS. Сравнение результатов показало приемлемое совпадение.

В то же время благодаря автоматизации занесения точек с глубинами и интерактивному отображению двумерного цветового поля с использованием ОВД-интерполяции в процессе оцифровки:

- существенно уменьшена трудоёмкость оцифровки;
- улучшено качество получаемого двумерного поля за счёт увеличения количества и более равномерного расположения точек;

•создаётся более гладкий рельеф за счёт ОВД-интерполяции не в узлы растра, а в клетки поля с задаваемыми размерами.

Для передачи результатов оцифровки в другие приложения формируется файл в формате GeoJSON.

Web-приложение в среде клиента, для разработки которого используется лишь «нативный» Javascript без каких-либо зависимостей от сторонних библиотек и сервисов, обладает рядом преимуществ за счёт выбора нескольких платформы для его размещения. Проверено, что такими платформами могут быть:

1) браузеры: Chrome, Firefox, Microsoft Edge и даже Android;
2) web-серверы под Apache ;
3) web-серверы под Node.js;
4) NW.js (node+webkit).

Первый вариант наиболее предпочтителен при ограниченности машинных ресурсов и может использоваться локально на бюджетных компьютерах и даже с использованием планшетов и смартфонов (под Android'ом).

Размещение на web-сервисах с серверами под Apache даёт возможность удалённого доступа к актуальной версии Web-приложения. Многие провайдеры размещают клиентские программы бесплатно или за очень небольшую плату.

Web-сервисы с Node.js могут быть предпочтительнее сервисов с Apache при желании развивать возможности Web-приложения за счёт доступа к средствам сервера на языке Javascript.

Вариант работы на платформе NW.js даёт возможность использовать Javascript в качестве системного языка с доступом к операционной системе локального компьютера, с соответствующими возможностями развития Web-приложения.

Список литературы

1. Бабин, А.В. Пространственный анализ данных в экологии и природопользовании. Лаб. практ.: учеб. пособие для ВУЗов. – СПб: РГГМУ, 2020. – 128 с.

2. Жуковский, О.И. Геоинформационная система QGIS: учебно-методическое пособие / О.И. Жуковский. – Томск, ТУСУР, 2018. – 81 с.

3. Калинин, А.А. Географические информационные системы: учеб. пособие/ А.А. Калинин, А.М. Бондаренко, Б.Н. Строгий, М.Н. Семенцов. – Зерноград: Азово-Черноморский инж. инст. ФГБОУ ВПО ДГАУ, 2015. – 58 с.

4. Лабберс, П. HTML 5 для профессионалов / П. Лабберс, Б. Олберс., Ф. Салим // - М.: ООО «И.Д. Вильямс», 2011. - 272 с.

5. Мальцев, К.А. Построение моделей пространственных переменных (с применением пакета Surfer): учеб. пособие / К.А. Мальцев, С.С. Мухарамова. – Казань: Казанский университет, 2014. – 103 с.

6. Матушкин, А.С. Картографирование и анализ пространственных данных с использованием геоинформационной системы QGIS: учеб. пособие / А.С. Матушкин. – Киров: ВятГУ, 2018. – 100 с.

7. Новикова, А.М. Сравнение возможностей интерполяционных модулей QGIS для морских климатических исследований при работе с массивом данных малой обеспеченности / А.М. Новикова, А.Б. Полонский, А.А. Новиков. – Материалы Международной конференции «ИнтерКарто. ИнтерГИС». Том 22 (2016), часть 1. С. 76-88.

8. Полупанов, В.Н. Имитационная модель загрязнения взвешенными веществами мелководных водных объектов / В.Н. Полупанов // сб. ст. по материалам XCV Международной научно-практической конференции «Инновационные подходы в современной науке». – № 11(95). – М., Изд. «Интернаука», 2021.

9. Сяо, Н. Алгоритмы ГИС / пер. с англ. А.А.Слинкина. – М.: ДМК Пресс, 2021. – 328 с.

10. Третьяков, В.Ю. Геоинформационные системы в экологии и природопользовании: программирование на Python в arcGIS и Quantum GIS/ . – СПб: РГГМУ, 2022. – 112 с.

11. Флэнаган Дэвид. Javascript. Полное руководство. 7-е изд. 2021. - 720 с.

12. Фримен, Э., Робсон Э. Изучаем HTML, XHTML и CSS. 2-е изд. - СПб.: Питер. 2014. - 720 с.

13. Фримен, Э., Робсон Э. Изучаем программирование на HTML5. — СПб.: Питер, 2013. — 640 с.

14. Хавербеке Марейн. Выразительный Javascript. Современное веб-программирование. 3-е изд. - СПб.: Питер, 2019. - 482 с.

15. Чернова, Е.Ю. Система координат 1942 года (СК-42): учебно-методическое пособие по курсу «Геоинформационные технологии» / Казанский гос. университет, 2002. – 25 с.

16. Lawhead, Joel. QGIS Python Programming Cookbook. Second Edition. 2017. ISBN 9781787124837. – 971 p.

17. Liu, Shu-Guang A study of two data grid interpolation algorithm based on Surfer Software / Shu-Guang Liu, Xi Chen, Shuhong Peng, Ying-Lian Ma, Jing Qian. International Conference on Machine Learning and Cybernetics, ICMLC 2010. – pp. 1045-1049.

18. Lou, Yiyong. "The acuity of volume analysis and data griding based on Surfer Software / Yiyong Lou. Liting Zhang. Zhuan Chen. Wei Yang. Mapping Sciences (in Chinese), 2009. - pp. 97-99

19. Shepard, D. A two-dimensional interpolation function for irregularly-spaced data / ACM National Conference. Harvard College Cambridge, Massachusetts, 1968). – pp. 517-524.

20. Документация QGIS 3.28. [Электронный ресурс]. URL: https://qgis.org/ru/docs/index.html (дата обращения 20.10.2023).

21. Геокалькулятор – пересчёт координат. [Электронный ресурс]. - URL: https://geoproj.ru/ .

22. ALGLIB - inverse distance weighting. [Электронный ресурс] – URL:

https://www.alglib.net/inverse-distance-weighting/ (дата обращения 20.10.2023)
23. ArcGIS 9. Spatial Analist / Руководство пользователя. [Электронный ресурс] – URL: https://reallib.org/reader?file=592955.
24. Beginners guide to the QGIS Field Calculator. [Электронный ресурс]. - URL: https://mapscaping.com/beginners-guide-to-the-qgis-field-calculator/ (дата обращения 20.10.2023).
25. GeoJSON. [Электронный ресурс]. – URL: https://geojson.org/ (дата обращения 20.10.2023).
26. Introducing JSON. [Электронный ресурс]. – URL: https://www.json.org/json-en.html (дата обращения 20.10.2023).
27. HIGHCHARTS [Электронный ресурс]. – URL: http://highcharts.com (дата обращения 20.10.2023).
28. NW.js [Электронный ресурс]. - URL: https://nwjs.io/ (дата обращения 20.10.2023).
29. NextGIS Web [Электронный ресурс]. - URL: https://nextgis.ru/blog/nextgis-frontend/ (дата обращения 20.10.2023).
30. OpenStreetMap [Электронный ресурс]. - URL: https:// www.openstreetmap.org/ (дата обращения 20.10.2023).
31. OpenLayers [Электронный ресурс]. - URL: https://openlayers.org/ (дата обращения 20.10.2023).
32. 8 Javascript-библиотек для визуализации данных в виде интерактивных карт [Электронный ресурс]. URL: https://habr.com/ru/articles/318600/ (дата обращения 20.10.2023).
33. 13 JavaScript Libraries to Create Interactive & Customized Maps [Электронный ресурс]. - URL: https://www.hongkiat.com/blog/javascript -libraries -for-interactive-maps/ (дата обращения 20.10.2023).
34. Как создавать визуализации и интерактивные карты на Javascript [Электронный ресурс]. - URL: https://qaa-engineer.ru/kak-sozdavat -vizualizaczii-i-nteraktivnye-karty-na-javascript/ (дата обращения 20.10.2023).